MCS-51 TRAINING BOARD

โดย
นายภิญโญ รัตนาพันธุ์
นาย ไมตรี ศรีตัมภวา
นายจิรศักดิ์ ธนิตาภิรมย์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต
ภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรม
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2534

ปริญญานิพนธ์ปีการศึกษา 2534

ภาควิชา เทคโนโลยีการวัดคุมทางอุตสาหกรรม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง

เรื่อง MCS-51 TRAINING BOARD

ผู้จัดทำ

1.นายภิญโญ	รัตนาพันธุ์	31.1202
2.นายไมตรี	ศรีตัมภวา	31.1212
3.นายจิรศักดิ์	ธนิตาภิรมย์	32.1440

อาจารย์ที่ปรึกษา

(อ.พิพัฒน์ เลาหสงคราม)

MCS-51 TRAINING BOARD

นายภิญโญ รัตนาพันธ์ 311202

นายไมตรี ศรีตัมภวา 311212

นายจิรศักดิ์ ธนิตาภิรมย์ 321440

อาจารย์ที่ปรึกษา

อาจารย์พิพัฒน์ เลาหสงคราม

ปีการศึกษา 2534

บทคัดย่อ

โครงงานเรื่อง MCS-51 TRAINING BOARD นี้ เป็นโครงงาน ที่มีจุดประสงค์ในการนำซีพียูตระกูล MCS-51 มาทดลองใช้งานในการสร้างเป็นซิงเกิลบอร์ด เพื่อนำไปรองรับการเขียนโปรแกรมขึ้นมาจัดการให้ทำงานตามต้องการ บอร์ดที่จัดทำขึ้นสามารถรับข้อมูลในรูปตัวอักษร ตัวอักษรพิเศษและตัวเลขได้ แล้วแสดงผลออกทางจอแอลซีดี ข้อมูลที่ป้อนเข้าไปจะประกอบอยู่ในรูปของโปรแกรมที่ใช้กับซีพียูตระกูล MCS-51 โดยเฉพาะ

MCS-51 TRAINING BOARD

MR PINYO RATTANAPAN 311202

MR MAITREE SRITAMBHWA 311212

MR JIRASAK TANITAPIROM 321440

ADVISOR

PIPAT LAOHASONGKRAM

SEMESTER 2534

ABSTRACT

THIS PROJECT, MCS-51 TRAINING BOARD,HAVE AN OBJECT TO BRING CPU,MCS-51 FAMILY,TO BUILD SINGLE BOARD AND WRITE MONITOR PROGRAM IN ORDER TO CONTROL IT. THIS SINGLE BOARD CAN BE INPUT BY ALPHABETS,MNEMONICS OR SPECIAL ALPHABETS. INSTRUCTION THEN DEMONSTRATED THE OUTPUT BY LCD. DATA THOSE INPUT ARE IN THE FORM OF PROGRAM THAT USE WITH CPU, MCS-51 FAMILY ONLY.

สารบัญ

บทนำ

โครงงานเรื่อง MCS-51 TRAINING BOARD นี้มีจุดมุ่งหมายในการทดลองนำเอาซีพียูในตระกูล MCS-51 มาใช้งาน ข้อดีของซีพียูตระกูลนี้ก็อยู่ที่เหมาะสำหรับการใช้ในการควบคุมกระบวนการต่างๆทางอุตสาหกรรมหรือการควบคุมอื่นๆแล้วแต่ว่าจะประยุกต์ใช้ในด้านใด ทั้งนี้ก็เพราะชิพซีพียูตระกูลนี้ถูกออกแบบมาให้สามารถจัดการประมวลผลได้ในระดับบิท อันเป็นการเหมาะสมอย่างยิ่งกับลักษณะของสัญญาณต่างๆที่เกิดมีขึ้นในการวัดและการควบคุม เช่น การทำงานจะอยู่ในสภาวะทำงานหรือว่าไม่ทำงาน หน่วยแสดงผลติดสว่างหรือว่าดับ เป็นต้น

แต่เนื่องจากโครงงานนี้เป็นการจัดทำบอร์ดที่ใช้ในการทดลอง มิใช่มุ่งเน้นไปที่การประยุกต์ใช้งาน จึงสามารถแสดงได้แต่เพียงโครงสร้างของระบบไมโครโปรเซสเซอร์โดยทั่วๆไป ลักษณะการจัดการของหน่วยอินพุทและเอาท์พุทที่ใช้ติดต่อกับโลกภายนอก และการเขียนมอนิเตอร์โปรแกรมขึ้นมาจัดการระบบทั้งหมดที่จัดทำขึ้นมา

หน่วยอินพุทหลักของบอร์ดก็คือ แผงคีย์บอร์ด มีจำนวนด้วยกันทั้งสิ้น 49 คีย์ ส่วนหน่วยเอาท์พุทหลักของบอร์ดก็คือ แผงแสดงผลที่อยู่ในรูปของจอแอลซีดี เหตุที่เลือกใช้จอแอลซีดีก็เพราะต้องการความชัดเจนในการแสดงผล เนื่องจากภาษาแอสเซมบลิของชิพไอซีตระกูลนี้มีการใช้สัญลักษณ์พิเศษที่จอแสดงผลประเภทเซกเมนต์ต่างๆแสดงได้ไม่ชัดเจนเท่าที่ควร

รายละเอียดต่างๆจะกล่าวถึงต่อไป

ระบบโดยทั่วไปของ MCS-51 TRAINING BOARD

โดยทั่วไปโครงสร้างพื้นฐานของคอมพิวเตอร์ จะประกอบไปด้วย

_หน่วยควบคุม (Control Unit)

_หน่วยความจำ (Memory Unit)

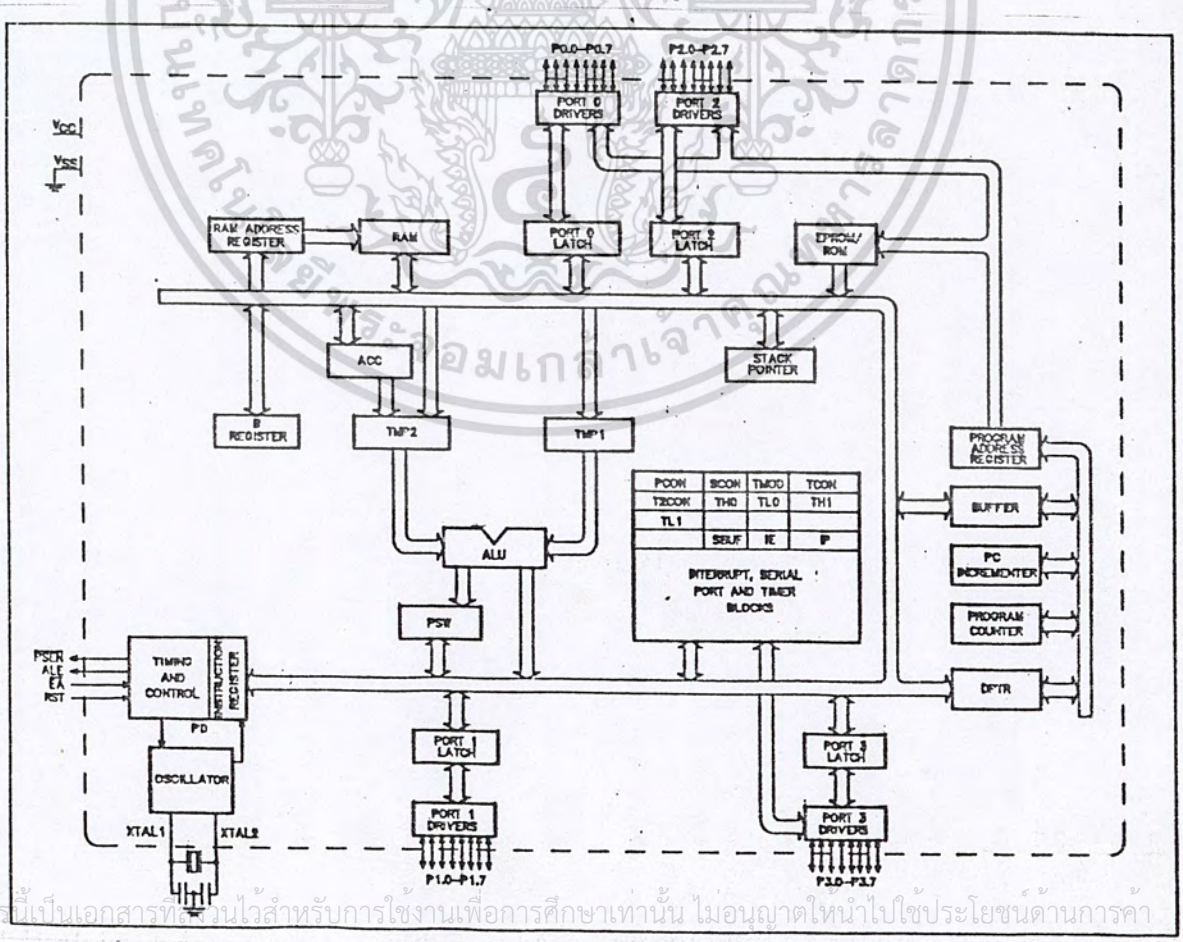
_หน่วยคำนวน (Arithmetic Unit)

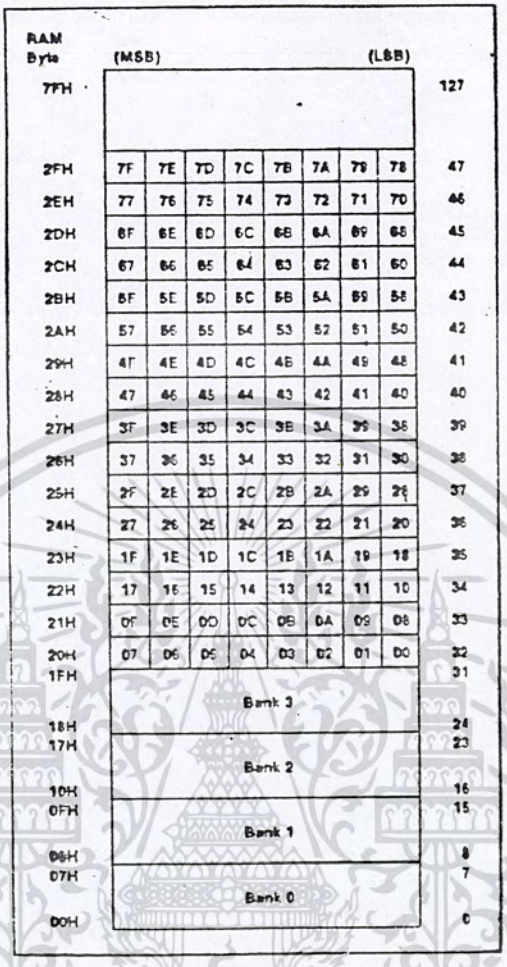
_หน่วยรับและส่งสัญญาณ (I/O Unit)

ซึ่งรายละเอียดของแต่ละหน่วยเป็นดังนี้

หน่วยควบคุมและหน่วยคำนวณ

จะถูกบรรจุอยู่ในชิพไอซี 8031 หรือซีพียูนั่นเอง





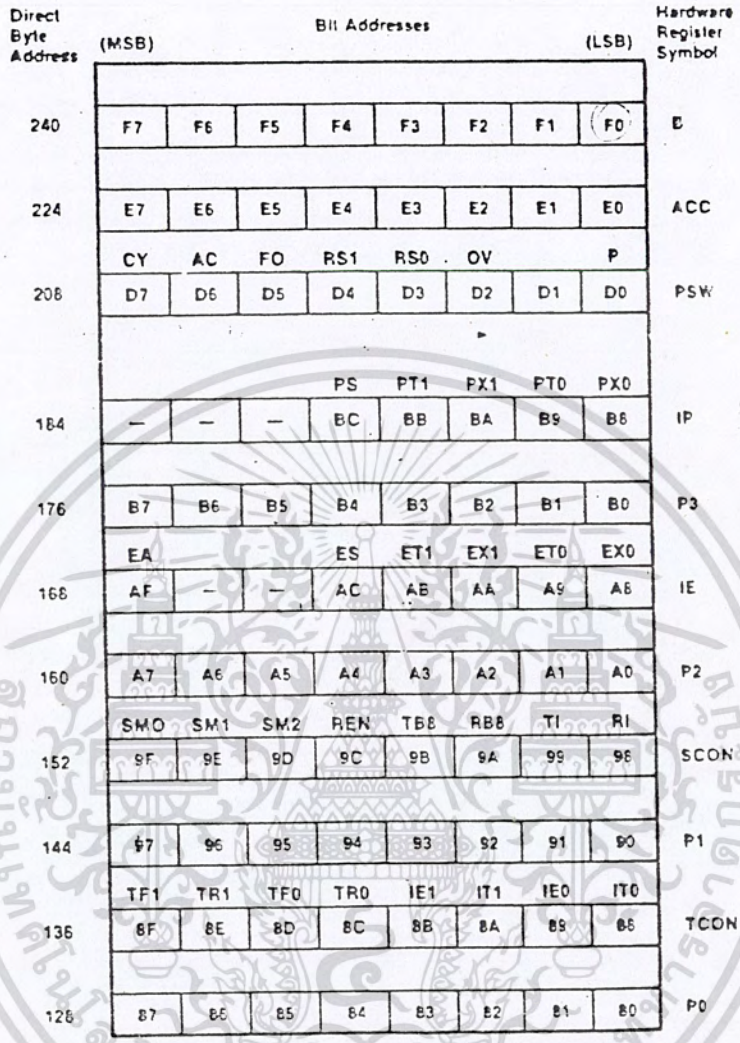
Special Function Register

มีรายละเอียดดังรูปหน้าถัดไป

รีจิสเตอร์ฟังค์ชันพิเศษแต่ละตัวมีรายละเอียดพอสังเขปดังนี้ คือ

@ แอคคิวมูเลเตอร์(Accumulator:ACC)

มีขนาด 8 บิท คำสั่งส่วนใหญ่จะอ้างอิงถึงรีจิสเตอร์ตัวนี้ โดยถือค่าภายในเป็นตัวตั้ง และรับค่าผลลัพธ์ที่ได้จากคำสั่งทางคณิตศาสตร์ เช่น บวก ลบ คูณ หาร เข้ามาเก็บไว้ ตัวแอคคิวมูเลเตอร์ยังสามารถใช้เป็นตัวกระทำหรือถูกกระทำในการทำงานทางตรรก และใช้เป็นตัวกลางในการถ่ายเทข้อมูลเมื่อต้องการติดต่อกับอุปกรณ์ภายนอก และหน่วยความจำภายนอก รวมถึงการตรวจสอบตารางข้อมูลด้วย



Direct Byte Address	(MSB)			Bit Addresses				(LSB)	Hardware Register Symbol
240	F7	F6	F5	F4	F3	F2	F1	F0	B
224	E7	E6	E5	E4	E3	E2	E1	E0	ACC
	CY	AC	F0	RS1	RS0	OV		P	
208	D7	D6	D5	D4	D3	D2	D1	D0	PSW
				PS	PT1	PX1	PT0	PX0	
184	—	—	—	BC	BB	BA	B9	B8	IP
176	B7	B6	B5	B4	B3	B2	B1	B0	P3
	EA			ES	ET1	EX1	ET0	EX0	
168	AF	—	—	AC	AB	AA	A9	A8	IE
160	A7	A6	A5	A4	A3	A2	A1	A0	P2
	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	
152	9F	9E	9D	9C	9B	9A	99	98	SCON
144	97	96	95	94	93	92	91	90	P1
	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	
136	8F	8E	8D	8C	8B	8A	89	88	TCON
128	87	86	85	84	83	82	81	80	P0

@ รีจิสเตอร์ B

มักจะใช้งานเมื่อทำคำสั่งการคูณละหาร โดยใช้เป็นที่เก็บตัวคูณหรือหาร และเป็นที่เก็บผลลัพธ์ตัวที่ 2 หลังการคูณ และเศษหลังการหาร

@ รีจิสเตอร์แสดงสถานะโปรแกรม(Program Status Word:PSW)

เป็นตัวที่ใช้ในการแสดงผลที่ได้หลังจากการใช้คำสั่งต่างๆ และเป็นตัวเลือกกลุ่มการทำงานของรีจิสเตอร์กลุ่มต่างๆ

@ ตัวชี้สแต็ก(Stack Pointer:SP)

มีขนาด 8 บิท จะเพิ่มค่าขึ้นโดยอัตโนมัติก่อนที่จะนำข้อมูลมาเก็บในหน่วยความจำระหว่างการใช้คำสั่ง PUSH และ CALL และจะลดค่าลงหลังจากที่ได้ถ่ายเทข้อมูลออกไปแล้วโดยคำสั่ง POP และ RETURN SPจะเริ่มที่ตำแหน่ง 07H ดังนั้น SP จะเริ่มบรรจุข้อมูลที่ตำแหน่ง 08H เป็นต้นไป

@ ตัวชี้ข้อมูล(Data Pointer:DPTR)

เป็นรีจิสเตอร์ขนาด 16 บิทที่ประกอบด้วยไบท์สูง (DPH) และไบท์ต่ำ(DPL) ที่สามารถแบ่งเป็นรีจิสเตอร์ขนาด 8 บิท 2 ตัวใช้งานได้อย่างอิสระหรือจะใช้รวมกันทั้ง 16 บิทก็ได้ในการ Increment หรือ Decrement เพื่อประโยชน์ในการใช้เป็นฐานของเลขที่อยู่ในรีจิสเตอร์ในการกระโดดโดยทางอ้อมในการใช้คำสั่งเกี่ยวกับตารางข้อมูลและชี้ตำแหน่งของหน่วยความจำภายนอก

@ พอร์ต 0 ถึง 3

รีจิสเตอร์ P0,P1,P2 และ P3 ของกลุ่มรีจิสเตอร์ฟังก์ชันพิเศษจะเป็นรีจิสเตอร์ที่ใช้ในการแล็ทช์ค่าของพอร์ท 0,1,2 และ 3 ตามลำดับ ในขณะใช้งาน

@ บัฟเฟอร์ข้อมูลอนุกรม(Serial Data Buffer:SBUF)

แบ่งออกเป็นรีจิสเตอร์ 2 ตัว เป็นบัฟเฟอร์ในการส่งตัวหนึ่งส่วนอีกตัวหนึ่งใช้ในการรับ เมื่อข้อมูลถ่ายเทเข้า SBUF จะทำการถ่ายเทเข้าบัฟเฟอร์ส่ง ซึ่งเป็นตัวจัดการข้อมูลอนุกรม วิธีการเคลื่อนย้ายเข้า SBUF ขึ้นอยู่กับการเริ่มแรกการส่ง เมื่อข้อมูลย้ายออกจาก SBUF จะเป็นการรับข้อมูลจากบัฟเฟอร์ตัวรับ

@ รีจิสเตอร์แคปเจอร์(Capture Register)

มีอยู่เป็นคู่ คือ RCAP2H และ RCAP2L สำหรับตัวตั้งเวลาหมายเลข 2 ในโหมดการใช้งานของรีจิสเตอร์ตัวนี้จะรับการเปลี่ยนแปลงที่เข้ามาที่ขา T2EX ตัว TH2 และ TL2 จะลอกข้อมูลเข้าไปในรีจิสเตอร์คู่ RCAP2H และ RCAP2L ด้วยการใช้ตัวตั้งเวลา จะมีโหมดการบรรจุอัตโนมัติขนาด 16 บิท สำหรับการใช้ตัวตั้งเวลา/ตัวนับ2

@ รีจิสเตอร์ควบคุม(Control Register)

กลุ่มรีจิสเตอร์ฟังก์ชันพิเศษ เช่น IP, IE, TMOD, TCON, T2CON, SCON และ PCON จะประกอบด้วยบิทที่ใช้ในการควบคุม และแสดงสถานะของการใช้งานในระบบอินเตอร์รัพ ตัวตั้งเวลา/ตัวนับ และพอร์ทอนุกรม

การจัดแบ่งตำแหน่งของ SFR เป็นดังนี้

รีจิสเตอร์	หน้าที่	ตำแหน่ง
ACC	Accumulator	
B	รีจิสเตอร์ B	
PSW	รีจิสเตอร์แสดงสถานะ	
SP	ตัวชี้สแต็ก	
DPTR	ตัวชี้ข้อมูล ประกอบด้วย DPH และ DPL	
P0	พอร์ท 0	
P1	พอร์ท 1	
P2	พอร์ท 2	
P3	พอร์ท 3	
IP	ตัวควบคุมการอินเตอร์รัพตามลำดับ	
IE	ตัวควบคุมการอินเตอร์รัพอีเนเบิ้ล	
TMOD	ตัวควบคุมตัวเลือกโหมดตัวตั้งเวลา/ตัวนับ	
TCON	ตัวควบคุมตัวตั้งเวลา/ตัวนับ	
T2CON	ตัวควบคุมตัวตั้งเวลา/ตัวนับ 2	
TH0	รีจิสเตอร์ตัวตั้งเวลา/ตัวนับ 0 (ไบท์สูง)	
TL0	รีจิสเตอร์ตัวตั้งเวลา/ตัวนับ 0 (ไบท์ต่ำ)	
TH1	รีจิสเตอร์ตัวตั้งเวลา/ตัวนับ 1 (ไบท์สูง)	
TL1	รีจิสเตอร์ตัวตั้งเวลา/ตัวนับ 1 (ไบท์ต่ำ)	

TH2	รีจิสเตอร์ตัวตั้งเวลา/ตัวนับ 2 (ไบท์สูง)
TL2	รีจิสเตอร์ตัวตั้งเวลา/ตัวนับ 2 (ไบท์ต่ำ)
RLDH	รีจิสเตอร์ตัวตั้งเวลา/ตัวนับ 2 ประจุใหม่อัตโนมัติ(ไบท์สูง)
RLDL	รีจิสเตอร์ตัวตั้งเวลา/ตัวนับ 2 ประจุใหม่อัตโนมัติ(ไบท์สูง)
SCON	ควบคุมการส่งข้อมูลอนุกรม
SBUF	บัฟเฟอร์การส่งข้อมูลอนุกรม
PCON	ควบคุมการใช้พลังงาน

<u>หน่วยความจำภายนอก</u>

แบ่งประเภทของการจัดการได้ 2 รูปแบบ คือ

@ การเข้าถึงหน่วยความจำโปรแกรมภายนอก

@ การเข้าถึงหน่วยความจำข้อมูลภายนอก

การเข้าถึงหน่วยความจำโปรแกรมภายนอกจะใช้สัญญาณ PSEN (Program Strobe Enable) แอคทีฟต่ำเป็นสัญญาณควบคุมการอ่าน ส่วนการเข้าถึงหน่วยความจำข้อมูลภายนอก จะใช้สัญญาณ RD และ WR แอคทีฟต่ำเป็นตัวควบคุม

การเฟ็ทโปรแกรมภายนอกจะใช้ขาแอดเดรส 16 บิท เสมอ ส่วนการเข้าถึงหน่วยความจำข้อมูลสามารถใช้ได้ทั้ง 16 บิทแอดเดรส เช่นคำสั่ง MOVX @DPTR

และ 8 บิทแอดเดรส เช่นคำสั่ง MOVX @Ri

เมื่อใช้ 16 บิทแอดเดรส ไบท์สูงของค่าแอดเดรสจะส่งออกที่พอร์ท 2 และจะคงสถานะค่านั้นตลอดช่วงวัฏจักรการอ่านและเขียน ตัวแสท์ของพอร์ท 2 ใน SFR จะต้องไม่ประกอบด้วยค่า "1"และค่าของข้อมูลใน SFR จะไม่มีการ set ถ้าช่วงวัฏจักรการใช้หน่วยความจำภายนอกไม่มีการเข้าถึงข้อมูลในวัฏจักรต่อมา ค่า SFRในพอร์ท 2 จะปรากฏค่าเดิมกลับมาใหม่ในวัฏจักรต่อมา

ถ้าใช้เป็น 8 บิทแอดเดรส ค่าใน SFR จะยังคงค่าเดิมที่ขาพอร์ต 2 ตลอดช่วงวัฏจักรการใช้ความจำภายนอก ซึ่งลักษณะนี้ จะเป็นการใช้งานด้านเพจของหน่วยความจำ

ในกรณีการใช้แอดเดรสไบท์ต่ำเป็นช่วงเวลามัลติเพลกซ์ กับข้อมูลของพอร์ท 0 ขาสัญญาณแอดเดรสและข้อมูลจะขับการ FET ทั้งสองตัวในพอร์ท 0 ให้เป็นบัฟเฟอร์ส่งข้อมูลออก ดังนั้นในระหว่างการใช้งานพอร์ต 0 จะไม่มีการรับกระแสเข้า จึงไม่จำเป็นต้องพูลอัพจากภายนอก สัญญาณ ALE: ADDRESS LATCH ENABBLE ก็จะใช้เป็นขาควบคุมการรับไบท์แอดเดรสเก็บไว้ภายนอก ซึ่งค่าแอดเดรสจะคงที่ที่ช่วงขอบขาลง ALE ดังนั้น ในวัฏจักรการเขียน ข้อมูลจะถูกเขียนออกไปที่พอร์ทก่อนที่ WR จะแอกทีฟต่ำ ส่วนวัฏจักรการอ่าน ข้อมูลจะรับเข้ามาที่พอร์ท 0 ก่อนที่สโตรบการอ่านจะปรากฏเล็กน้อย และระหว่าง การเข้าถึงหน่วยความจำภายนอก ตัวซีพียูจะส่งค่า 0FFH มาเก็บไว้ที่พอร์ต 0 ของ SFR

การใช้หน่วยความจำโปรแกรมภายนอก จะขึ้นอยู่กับสองกรณี คือ

1. เมื่อไรก็ตามที่ EA แอกทีฟ หรือ
2. เมื่อไรก็ตามที่ตัวนับโปรแกรม ประกอบด้วยตัวเลขที่มีค่ามากกว่า 0FFH (และ 1FFH สำหรับ 8052)

ในรุ่นที่ไม่มีรอม ในตัว ให้ใช้ค่าแอกทีฟต่ำป้อนเข้าที่ขา EA เมื่อกำหนดการเฟทช์โปรแกรมภายนอกที่มีขนาดต่ำกว่า 4 กิโลไบท์ได้

เมื่อโปรแกรมภายนอกถูกใช้งาน ทั้ง 8 ขาของพอร์ท 2 จะทำการส่งค่าแอดเดรสออกมาด้วย ทำให้ไม่สามารถจะใช้งานเป็นพอร์ตไอโอได้ ในระหว่างการเฟทช์โปรแกรมภายนอก เพราะจะส่งค่าไบท์สูงจาก PC ออกมาที่พอร์ต 2 นี้ และในระหว่างการเข้าถึงข้อมูลภายนอก จะใช้พอร์ท 2เป็นตัวส่งค่าแอดเดรสไบท์สูงจาก DPH ใน SFR ขึ้นอยู่กับการใช้คำสั่งว่าใช้แบบให้คำสั่งส่งเอาท์พุทออกจาก DPH ในการกำหนดแอดเดรสข้อมูลภายนอกก็จะใช้คำสั่ง MOVX @DPTR หรือจะใช้แบบให้ข้อมูลส่งข้อมูลออกที่พอร์ทของ SFR ก็จะใช้คำสั่ง MOVX @Ri

สัญญาณที่เกี่ยวข้องกับการติดต่อหน่วยความจำโปรแกรมภายนอก

และหน่วยความจำข้อมูลภายนอก คือ

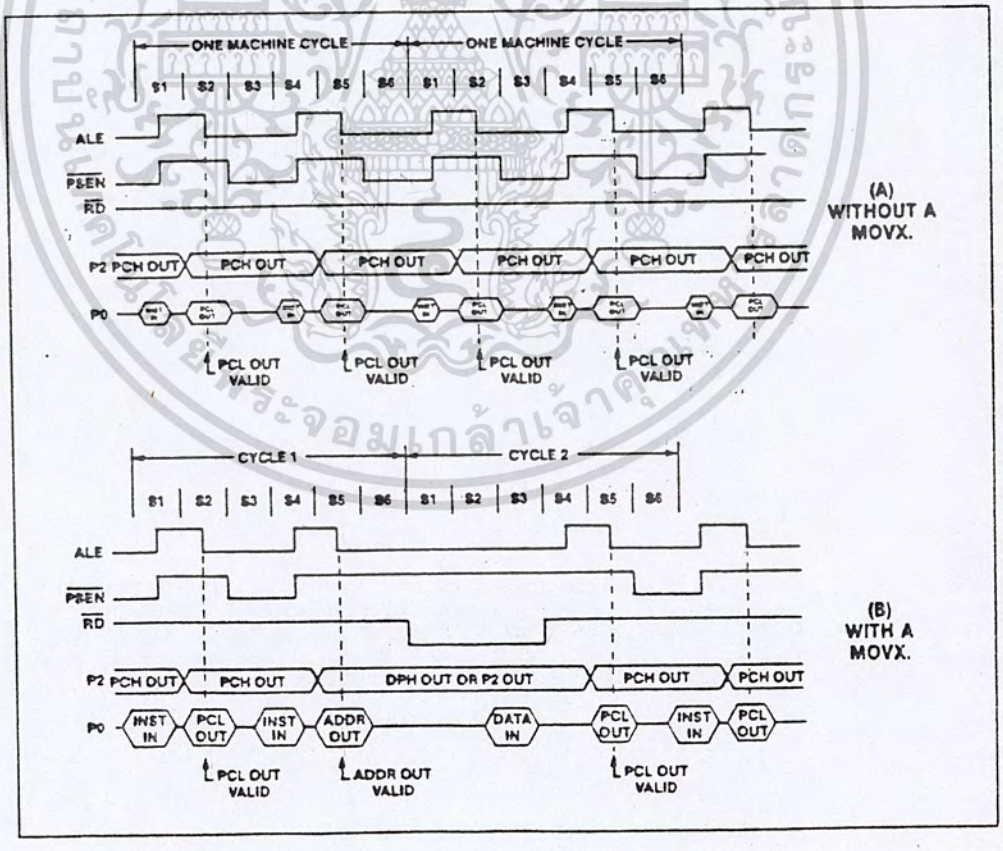
สัญญาณ PSEN

และ ALE

รายละเอียดของแต่ละสัญญาณจะเป็นดังนี้

สัญญาณ $\overline{\text{PSEN}}$

ใช้ในการควบคุมการเฟทช์อ่านโปรแกรมจากภายนอก $\overline{\text{PSEN}}$ จะไม่แอ็คทีฟถ้ามีการเฟทช์โปรแกรมภายใน เมื่อซีพียูมีการเข้าถึงการใช้โปรแกรมภายนอก PSEN จะแอ็คทีฟ 2 ครั้งในแต่ละช่วงวัฏจักรการเฟทช์ ยกเว้นคำสั่ง MOVX ช่วงเวลาของการที่ $\overline{\text{PSEN}}$ เกิดการแอ็คทีฟจะไม่เหมือนกับช่วงที่ RD เกิดการแอ็คทีฟ ช่วงวัฏจักรการอ่านที่สมบูรณ์จะรวมเอาช่วงที่ ALE แอ็คทีฟ และแอ็คทีฟซ้ำลูกที่สองและสัญญาณควบคุม RD ที่เกิดพัลส์ต่ำประกอบเข้าด้วยกัน ซึ่งจะใช้เวลา 12 สถานะคาบเวลา ส่วนช่วงเวลาของ $\overline{\text{PSEN}}$ ที่สมบูรณ์ จะรวมเอาช่วงเวลาที่ ALE แอ็คทีฟและแอ็คทีฟซ้ำลูกที่สองประกอบเข้าด้วยกันกับสัญญาณควบคุม $\overline{\text{PSEN}}$ ซึ่งจะใช้เวลา 6 สถานะคาบเวลา ลักษณะการทำงานตามลำดับของวัฏจักรการอ่านทั้ง 2 แบบ แสดงดังรูป



สัญญาณ ALE

ฟังก์ชันหลักของ ALE คือ การใช้งานในการให้จังหวะที่แน่นอนในการแลทช์เอาไบท์ต่ำของแอดเดรสจาก P_0 ไปเก็บไว้ภายนอก เพื่อใช้ในการถอดรหัสแอดเดรสโปรกรมภายนอก โดยจะให้ ALE ทำงานแอ็คทีฟสองครั้งในทุก ๆวัฏจักรแมชีน สัญญาณนี้จะเกิดขึ้นตลอดแม้ว่าจะไม่ได้เฟทช์จากภายนอก มีเพียงช่วงเวลาเดียวเท่านั้นที่ ALEไม่เกิดพัลส์ คือ ระหว่างการเข้าถึงหน่วยความจำภายนอก ตามรูปที่ผ่านมาจะเห็นว่าพัลส์แรกของ ALE ในวัฏจักรที่สองของคำสั่ง MOVX ขาดหายไป หรือมีเพียงพัลส์เดียวในหนึ่งคำสั่ง ลักษณะของพัลส์ที่เกิดขึ้นจะคงที่ในอัตรา 1/6 เท่าของสัญญาณความถี่ของออสซิลเลเตอร์ และยังสามารถนำมาใช้เป็นสัญญาณนาฬิกาภายนอกหรือกำหนดเวลาได้

การใช้เนื้อที่ของโปรแกรมภายนอกซ้อนกับหน่วยความจำข้อมูล

การใช้งานบางครั้ง อาจจะทำงานตามโปรแกรมจากหน่วยความจำที่มีลักษณะต่ำแหน่งซ้อนกันกับการใช้ข้อมูล ใน MCS-51 ตัวโปรแกรมภายนอกและหน่วยความจำข้อมูลสามารถที่จะทำงานร่วมกันได้ด้วยการให้สัญญาณ $\overline{\text{PSEN}}$ และ RD เข้า AND เกตกันก่อน จะเห็นว่าที่แอ็คทีฟของการ AND ที่เกิดจากสัญญาณทั้งสอง จะให้สัญญาณสโตรบการอ่านแอ็คทีฟต่ำในช่วงการอ่าน และแอ็คทีฟสูงในช่วงการเฟทช์ ซึ่งทำให้สามารถใช้ตำแหน่งหน่วยความจำตำแหน่งเดียวกันได้ เนื่องจากวัฏจักร $\overline{\text{PSEN}}$ จะเกิดขึ้นเร็วกว่าวัฏจักรการอ่าน ($\overline{\text{RD}}$) ดังนั้น ความเร็วของหน่วยความจำโปรแกรมต้องเร็วเพียงพอที่จะให้ทำให้วัฏจักร $\overline{\text{PSEN}}$ ทำงานได้อย่างสมบูรณ์

ต่อไปจะได้กล่าวถึงการนำเอาสัญญาณต่างๆเหล่านี้ไปใช้งาน

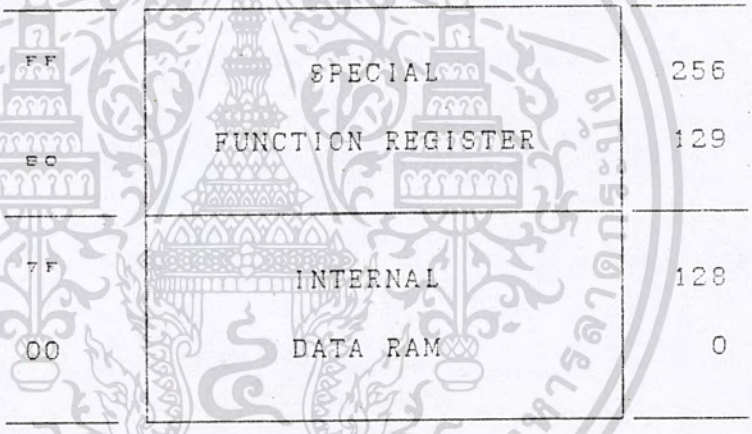
หน่วยความจำ (MEMORY UNIT)

สถาปัตยกรรมของ MCS-51 ได้แบ่งหน่วยความจำมาให้บนชิพ พร้อมทั้งสามารถที่จะขยายหน่วยความจำภายนอกได้ ซึ่งการแบ่งหน่วยความจำจะประกอบด้วย 3 ส่วนที่สำคัญ คือ

256	ไบท์	หน่วยความจำข้อมูลภายใน
64	กิโลไบท์	หน่วยความจำข้อมูลภายนอก
64	กิโลไบท์	หน่วยความจำโปรแกรม

หน่วยความจำข้อมูลภายใน

แบ่งตามลักษณะงาน คือ



Internal Data Ram

มีรายละเอียดแสดงดังรูปหน้าถัดไป

บริเวณล่างของแรม ตั้งแต่ตำแหน่งที่ 0_31 ประกอบด้วย 4 แบงค์ แต่ละแบงค์มีรีจิสเตอร์ 8 ตัว แบงค์เหล่านี้จะถูกเรียกใช้ให้อีเนเบิ้ลได้คราวละหนึ่งแบงค์เท่านั้น ด้วยการกำหนดเริ่มแรกภายใน 2 บิทของรีจิสเตอร์ PSW ว่าจะเลือกใช้แบงค์ใดภายใน 4 แบงค์เกิดการอีเนเบิ้ล

ส่วนบริเวณตั้งแต่ 20H ถึง 2FH จำนวน 16 ตำแหน่งๆละ 1 ไบท์ สามารถกำหนดแอดเดรสของแต่ละบิทได้

การอ่านโปรแกรมและการทำงานตามคำสั่งโปรแกรมกระทำที่ส่วนนี้ด้วย การใช้ส่วนคณิตศาสตร์และตรรกศาสตร์จะทำงานร่วมกับรีจิสเตอร์ A, B, PSW,SP,PC และ DPTR โดยที่ตัวแปรขนาดต่างๆจะมีขนาด 8 บิต ที่มีลักษณะการทำงานทางคณิตศาสตร์เป็น บวก ลบ คูณ และหาร ส่วนการทำงานทางตรรกศาสตร์จะอยู่ในลักษณะของการ AND OR XOR รวมทั้งการเลื่อนและวนรอบบิต การเคลียร์ค่าและกลับค่าเป็นต้น หน่วยคณิตศาสตร์และตรรกศาสตร์ยังสามารถที่จะตัดสินใจในการให้กระโดดไปทำคำสั่งของโปรแกรมในส่วนอื่นๆตามเงื่อนไขที่ตั้งขึ้น และยังแบ่งรีจิสเตอร์ชั่วคราวใช้สำหรับเป็นทางผ่านชั่วคราวของข้อมูลในการถ่ายเทภายในระบบอีกด้วย

สิ่งสำคัญในการทำงานทางสถาปัตยกรรมของ MCS-51 คือ ความสามารถในการทำงานสำหรับข้อมูลขนาด 8 บิท และ 1 บิท การใช้งานในระดับบิทในการเซทเคลียร์หรือกลับค่า การเคลื่อนย้าย การทดสอบ และใช้ในการคำนวณทางตรรกขนาด 1 บิทความสามารถเช่นนี้เหมาะสำหรับใช้ในงานควบคุมของสัญญาณเข้าและออกที่มีการคิดและออกแบบทางตรรกด้วยพีชคณิตบูลลีน ซึ่งโดยปกติทำได้ลำบากสำหรับไมโครโปรเซสเซอร์ทั่วๆไป งานในลักษณะเช่นนี้จึงได้ชื่ออีกอย่างหนึ่งว่า ตัวประมวลผลบูลลีน (Boolean)

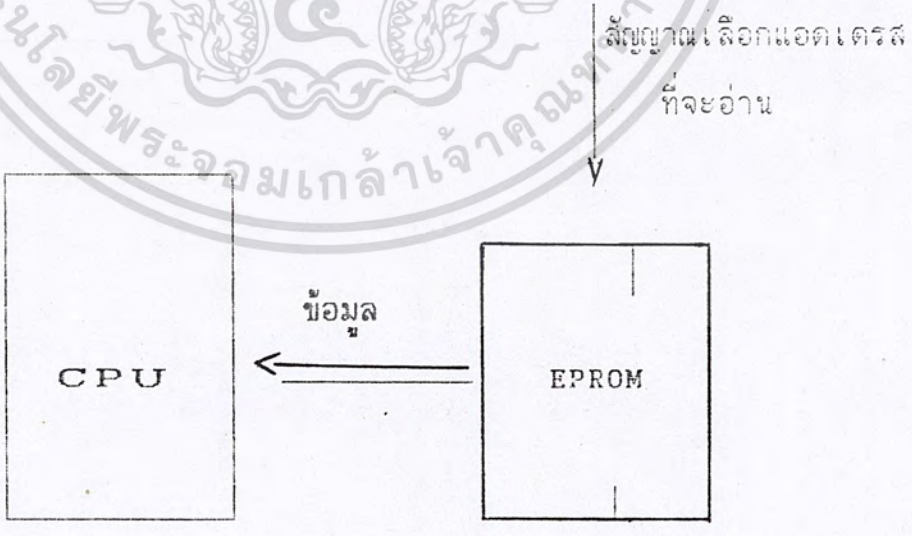
MCS-51 มีรีจิสเตอร์พิเศษซึ่งอยู่ในกลุ่มที่เรียกว่า Special Function Register ทำหน้าที่ควบคุมการทำงานลักษณะต่างๆ คือ

รีจิสเตอร์	หน้าที่
IP	ควบคุมการอินเทอร์รัพตามลำดับ
IE	ควบคุมการอินเทอร์รัพอิเนเบิ้ล
TMOD	ควบคุมการเลือกโหมดตัวตั้งเวลา/ตัวนับ
TCON	ควบคุมการตัวตั้งเวลา/ตัวนับ
T2CON	ควบคุมการตัวตั้งเวลา/ตัวนับ 2
SCON	ควบคุมการส่งข้อมูลอนุกรม
PCON	ควบคุมการใช้พลังงาน

การเชื่อมต่อกับ EPROM

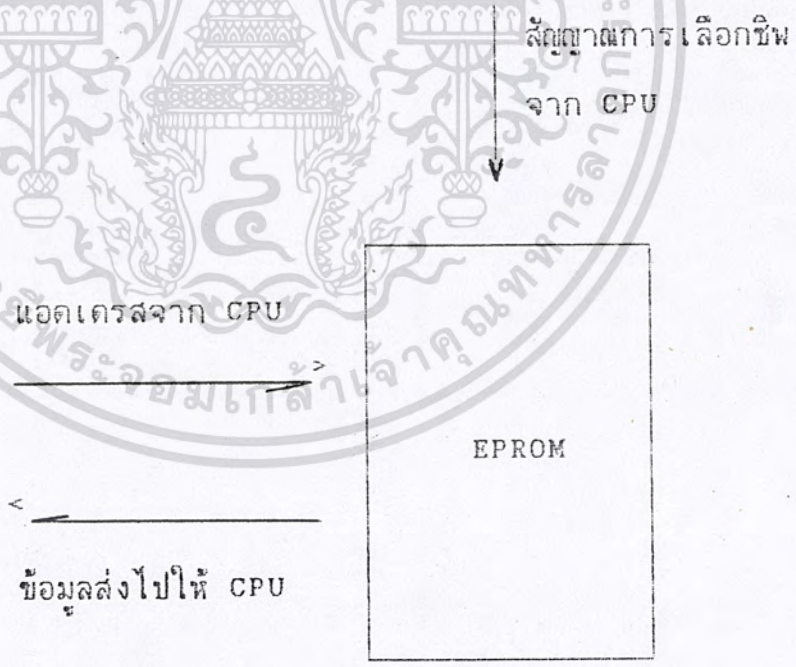
EPROM จะใช้ในการเก็บมอนิเตอร์โปรแกรมทั้งหมด EPROM สามารถหามาใช้ และทดลองได้โดยง่าย มีราคาถูก ง่ายต่อการต่อวงจร และเปลี่ยนแปลงโปรแกรม EPROM ที่ใช้ทั่วไปมีโครงสร้างเป็นไบท์ ซึ่งเราจะกล่าวถึงขนาดความจุของ EPROM เป็นจำนวนกิโลไบท์ เช่น 2048 เป็น EPROM ขนาด 2 กิโลไบท์ ซึ่งหมายถึง 2 x 1024 ไบท์ ในทางคอมพิวเตอร์ 1 กิโลไบท์ มีค่าเท่ากับ 2^{10} หรือ 1024 ไบท์ ทั้งนี้เพราะการอ้างอิงแอดเดรส จะอ้างอิงกันด้วยสายสัญญาณแอดเดรส เช่น ถ้าอ้างอิงด้วยแอดเดรส 10 บิต ก็จะอ้างอิงได้สูงสุด 2^{10} แอดเดรส หรือ 1 กิโลไบท์ นั่นเอง

การเชื่อมโยงกับ EPROM นี้ จึงเป็นลักษณะที่อ่านข้อมูลได้อย่างเดียวไม่สามารถเขียนได้ การอ่านนั้นจะอ่านเป็นคำสั่งมาทำงานตาม หรือจะอ่านข้อมูลมาประมวลผลในซีพียูก็ได้ แผนผังการเชื่อมโยงระหว่างซีพียู กับ EPROM แสดงได้ดังรูป



EPROM มีข้อดีคือ สามารถโปรแกรมและลบได้ง่าย การโปรแกรมจะใช้ขั้นตอนหรือวิธีการที่ไม่ยุ่งยาก การลบนั้นจะใช้วิธีฉายแสงอัลตราไวโอเลต นานประมาณ 5-10 นาที

เนื่องจาก EPROM เป็นหน่วยความจำที่มีความหนาแน่นสูง และสามารถอ่านได้เพียงอย่างเดียว ในหนึ่งชิพอาจจะมีความจุได้สูงถึง 32 กิโลไบท์ ซึ่งนับว่ามีขนาดของหน่วยความจำจำนวนมาก รูปข้างล่างจะแสดงให้เห็นส่วนประกอบพื้นฐานของ EPROM ซึ่งจะเห็นว่า มีสัญญาณต่าง ๆ ที่เกี่ยวข้องกับ EPROM และทุกชิ้นที่อยู่ใน EPROM มักจะมีการแบ่งแยกหน้าที่เสมอ เช่น ขาแอดเดรส ของ EPROM เป็นอินพุต ส่วนขาข้อมูลจะเป็นเอาท์พุท โดยหลักการแล้วขาข้อมูลจะต่อเข้ากับบัสข้อมูลซึ่งเป็นบัสสองทาง ดังนั้น เอาท์พุท ของ EPROM ในส่วนขาข้อมูลนี้ มักจะเป็นลอจิก 3 สถานะ ซึ่งถ้าไม่ใช้ก็จะอยู่ในสถานะที่มีอิมพีแดนซ์สูง



ขั้นตอนการอ่านข้อมูลจาก EPROM

ในการอ่านข้อมูลจาก EPROM นี้ จะต้องมีขั้นตอนเรียงลำดับกันไป ซึ่งปกติไมโครโปรเซสเซอร์ ได้จัดลำดับขั้นการอ่านข้อมูลไว้แล้ว ซึ่งมีรายละเอียดดังนี้

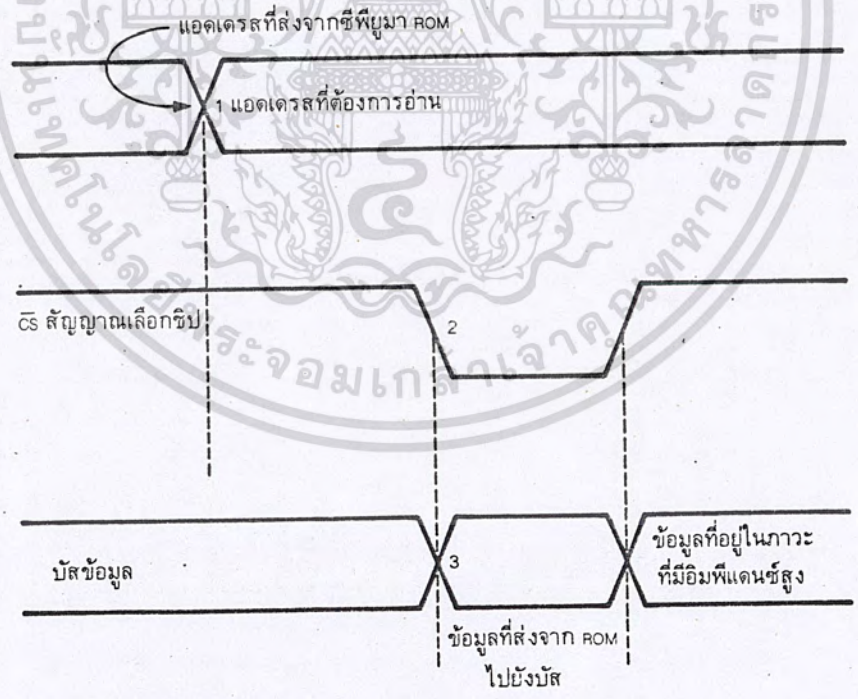
1. ซีพียู จะส่งแอดเดรส ไปให้ EPROM แอดเดรสดังกล่าวนี้จะปรากฏ เป็นแอดเดรสที่ต้องการจะอ่าน ใน EPROM

2. ซีพียูจะต้องใช้ช่วงเวลาของการส่งแอดเดรส ยาวนานพอประมาณ โดยปกติต้องประมาณ 100-300 นาโนวินาที มิฉะนั้น EPROM จะตอบสนองไม่ทัน

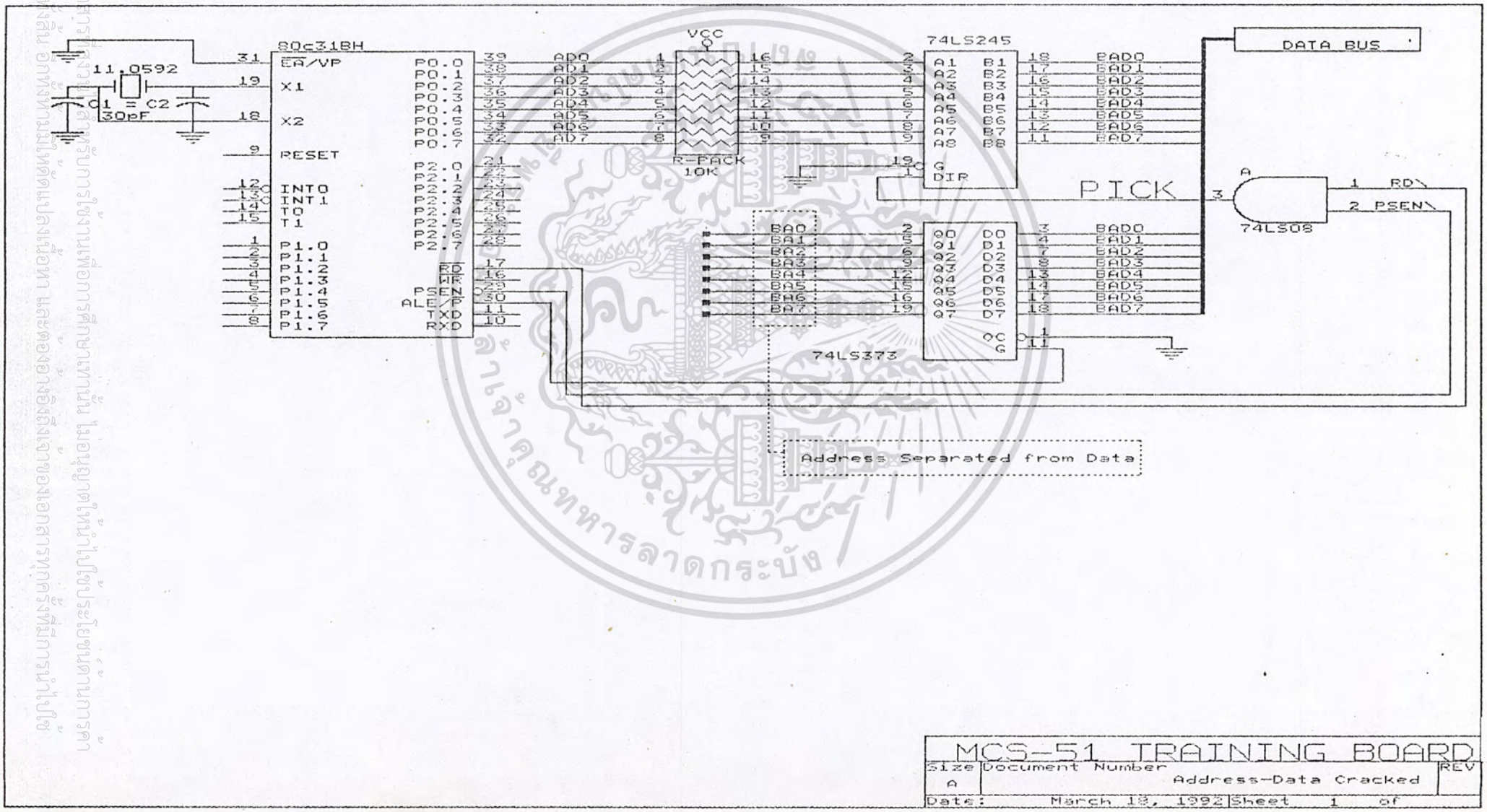
3. ซีพียูจะส่งสัญญาณ ไปทำการเลือก EPROM เรียกว่า สัญญาณ $\overline{CS}$ (CHIP SELECT) เพื่อบอกว่า ต้องการเลือก EPROM ซึ่งเป็นการส่งสัญญาณเพื่อยืนยันการเลือกชิป นั่นเอง

4. ข้อมูลจะผ่านออกทางขาข้อมูลชั่วขณะ จังหวะการเลือกชิป และเมื่อขาที่ใช้ในการเลือกชิปไม่แอ็คทีฟ ขาส่งข้อมูลก็จะเข้าสู่ภาวะที่มีอิมพีแดนซ์สูง

จากลักษณะดังกล่าว เขียนเป็นแผนผังเวลาออกมาได้ดังรูป



หมายเหตุ 1 หมายถึงการส่งแอดเดรส
2 หมายถึงการกำหนดสัญญาณ $\overline{CS}$
3 หมายถึงการอ่านข้อมูล



MCS-51 TRAINING BOARD		
Size: A	Document Number: Address-Data Cracked	REV
Date: March 18, 1992	Sheet 1 of	

การเชื่อมต่อ EPROM เข้ากับบัส

สามารถเชื่อม DATA BUS เข้าโดยตรงกับ DATA BUS ของซีพียู เลย แต่ในส่วนของ แอดเดรส 8 บิตล่าง จะต้องนำไปผ่าน 74LS373 เสียก่อน เพื่อทำการแยกข้อมูลและแอดเดรสที่ปนกันมาออกจากกัน ดังรูป

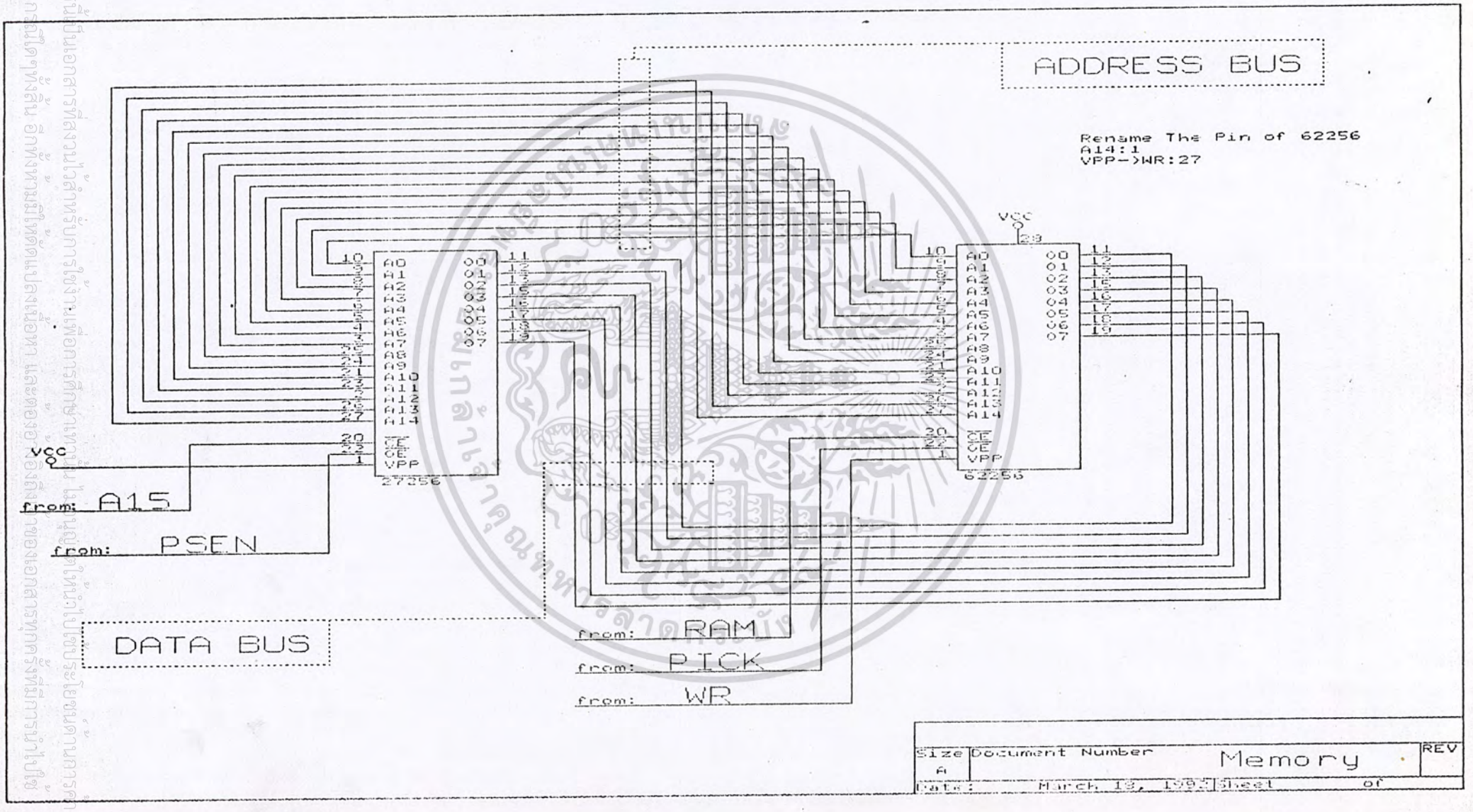
สัญญาณที่ควบคุมการแยกคือ ALE (ADDRESS LATCH ENABLE) ซึ่งมีลักษณะการทำงานดังที่กล่าวมาก่อนหน้านี้เช่นกัน เมื่อทำการแยกแอดเดรส A_7 - A_0 ออกจากข้อมูลได้แล้วก็ทำการเชื่อมเข้าขาแอดเดรสของ EPROM ได้ต่อไป

ขา $\overline{CE}$ (CHIP ENABLE) แอ็คทีฟที่ "0" จึงทำการนำเอาสัญญาณ A_{15} มาต่อเข้าที่ขานี้ ทั้งนี้จากหลักการที่ว่า EPROM เบอร์ 27256 นี้มีขนาด 32 กิโลไบท์ สามารถจัดการให้อยู่ในแอดเดรส ตั้งแต่ 0000H-7FFFH ซึ่งแอดเดรสช่วงนี้มีลักษณะดังนี้

A_{15} A_{14} A_{13} A_{12}	A_{11} — A_0	เลขฐาน 16
0 0 0 0	X	0 X X X
0 0 0 1	X	1 X X X
0 0 1 0	X	2 X X X
0 0 1 1	X	3 X X X
0 1 0 0	X	4 X X X
0 1 0 1	X	5 X X X
0 1 1 0	X	6 X X X
0 1 1 1	X	7 X X X

จะเห็นได้ว่า A_{15} จะมีสถานะ "0" ตลอดช่วงแอดเดรส จึงใช้ A_{15} มาควบคุมการทำงานของ EPROM ได้

ขา $\overline{OE}$ (OUTPUT ENABLE) นำเอาสัญญาณ $\overline{PSEN}$ มาต่อเข้าที่ขานี้โดยอาศัยหลักการที่กล่าวในเรื่อง $\overline{PSEN}$

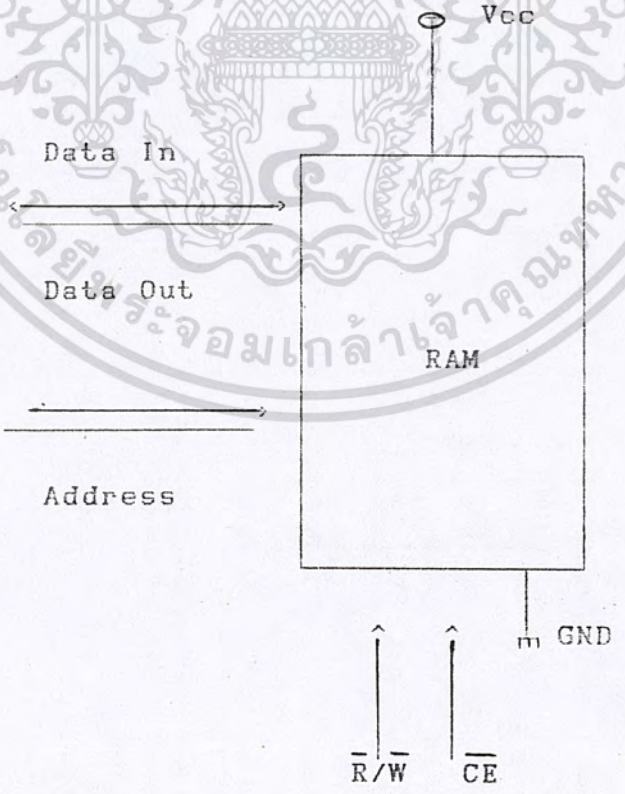

ADDRESS BUS
Rename The Pin of 62256
A14:1
VPP->WR:27
VCC
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
O0
O1
O2
O3
O4
O5
O6
O7
CE
OE
VPP
27256
62256
from: A15
from: PSEN
DATA BUS
from: RAM
from: PICK
from: WR
Size A
Document Number
Memory
REV
Date: March 13, 1992
Sheet of

การเชื่อมต่อกับ RAM

หน่วยความจำที่ต่อเข้ากับระบบไมโครคอมพิวเตอร์ที่สำคัญอีกส่วนหนึ่งก็คือ RAM ซึ่งเป็นชิพไอซีที่ทำหน้าที่เป็นพื้นที่สำหรับการนำข้อมูลที่ต้องการประมวลผลมาเก็บไว้ชั่วคราว โดยทั่วไปแล้ว RAM จึงเป็นไอซีที่ใช้ในการเก็บข้อมูล และสามารถเขียนข้อมูลใหม่ทับลงไปได้ง่าย แต่ถ้าไฟฟ้าดับ ข้อมูลก็จะสูญหายไปทันที

การเชื่อมโยงกับ RAM เป็นเรื่องที่ทำได้ง่าย เพราะโครงสร้างของซีพียู ได้รับการออกแบบให้มีสายสัญญาณ และจังหวะการเขียนหรืออ่านที่พอดีกับการเชื่อมต่อกับ RAM อยู่แล้ว สำหรับบอร์ดที่ทำขึ้นมานี้ใช้ RAM เบอร์ 62256

เพื่อให้เข้าใจเกี่ยวกับการเชื่อมต่อ RAM เข้ากับระบบไมโครโปรเซสเซอร์ จึงจำเป็นจะต้องเข้าใจระบบพื้นฐานทั่วไปของ RAM เสียก่อน ซึ่งสามารถเขียนเป็นแผนผังได้ดังรูป



จากรูปเป็นโครงสร้างพื้นฐานของ RAM ซึ่งมีส่วนของสัญญาณที่เกี่ยวข้องกับการควบคุมอยู่หลายเส้น สัญญาณที่สำคัญประกอบด้วย

1. D_{in} เป็นสายสัญญาณที่จะนำข้อมูลเข้าไปเก็บใน RAM เมื่อมีการเขียนข้อมูล

2. D_{out} เป็นสายสัญญาณที่จะนำข้อมูลออกจาก RAM เมื่อข้อมูลได้รับการอ่านเรียบร้อยแล้ว

3. แอดเดรส เป็นสายสัญญาณที่ทำหน้าที่กำหนดแอดเดรสของ RAM เพื่อการเขียนหรือการอ่าน

4. $\overline{R}/\overline{W}$ เป็นสายสัญญาณที่ทำหน้าที่กำหนดการเขียนหรือการอ่านข้อมูลบน RAM

5. $\overline{CE}$ เป็นสายสัญญาณเพื่อเลือกชิปในกรณีที่ต้องการต่อหลาย ๆ ชิปในระบบเพื่อที่จะได้ทราบว่า ชิปใดได้รับการเลือก

6. V_{cc}, GND เป็นสายไฟเลี้ยงวงจรและกราวด์

การทำงานของ RAM จะต้องใช้สัญญาณต่าง ๆ เหล่านี้ร่วมกัน โดยสัญญาณควบคุมนี้จะได้รับการสร้างขึ้นมาจากไมโครโปรเซสเซอร์ แต่อาจจะต้องมีการปรุงแต่งสัญญาณอีกบ้างเล็กน้อย

ได้กล่าวถึงการเชื่อมต่อระหว่างไมโครโปรเซสเซอร์ กับ EPROM ไปแล้ว สำหรับการเชื่อมต่อระหว่างไมโครโปรเซสเซอร์กับ RAM อาจมีความแตกต่างกันอยู่บ้าง ซึ่ง RAM ที่ใช้จะบอกขนาดความจุเป็น 1024 x 4 หรือ 1 KB x 4 ซึ่งหมายถึง ความสามารถเก็บข้อมูลได้ทั้งสิ้น 1024 ตำแหน่ง ตำแหน่งละ 4 บิต ดังนั้น RAM เบอร์ 6116 ที่บอกความจุไว้ 2 KB x 8 จะหมายถึง สามารถเก็บข้อมูลได้ 2048 ตำแหน่ง ตำแหน่งละ 8 บิต ปัจจุบันขนาดความจุของ RAM สูงขึ้น เช่น เบอร์ 6264 มีความจุ 8 KB x 8 และเบอร์ 62256 มีความจุ 32 KB x 8

การอ่านข้อมูลจาก RAM

สมมุติว่า ต้องการจะอ่านข้อมูลจาก RAM ที่มีข้อมูลพร้อมอยู่แล้วมาใช้งาน จะต้องมีลำดับการทำงานดังนี้

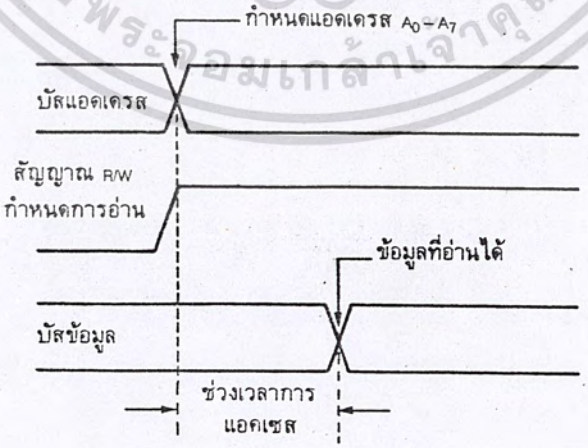
1. ขั้นแรก กำหนดแอดเดรสที่ต้องการอ่าน ให้กับ RAM วงจรภายใน RAM จะทำการถอดรหัสเพื่อกำหนดตำแหน่ง ที่แท้จริงสำหรับการอ่าน

2. กำหนด สัญญาณที่ขา R/W ให้ถูกต้องตามลอจิก โดยการอ่านหน่วยความจำบางชิป จะต้องกำหนดลอจิก "1" บางชิปอาจจะเป็นลอจิก "0" การกำหนดให้เป็นลอจิก "0" หรือ ลอจิก "1" จะต้องพิจารณาโดยดูจากข้อมูลของชิปนั้นๆ ประกอบด้วย

3. จะต้องให้ระบบรออยู่ชั่วขณะหนึ่ง ซึ่งเรียกช่วงเวลานี้ว่า ช่วงเวลาการอ่าน (read access time) ในการอ่านนี้จะใช้สัญญาณเลือกชิปทำการเลือกโดยส่งสัญญาณ CE มาก่อน

4. ในช่วงเวลาขณะที่รอนี้ข้อมูลที่ได้รับการอ่าน จะมาปรากฏที่สายสัญญาณ D_{out} เพื่อให้ไมโครโปรเซสเซอร์ รับข้อมูลออกไป

เมื่อเขียนเป็นแผนผังเวลา เพื่อแทนการทำงานทั้ง 4 ขั้นตอนที่กล่าวถึงนี้จะได้ดังรูป



· การเขียนข้อมูลใน RAM

การเขียนข้อมูลใน RAM มีลักษณะคล้ายกับการอ่าน แต่มีข้อแตกต่างอยู่บ้าง สำหรับลำดับการเขียนข้อมูลใน RAM มีดังนี้

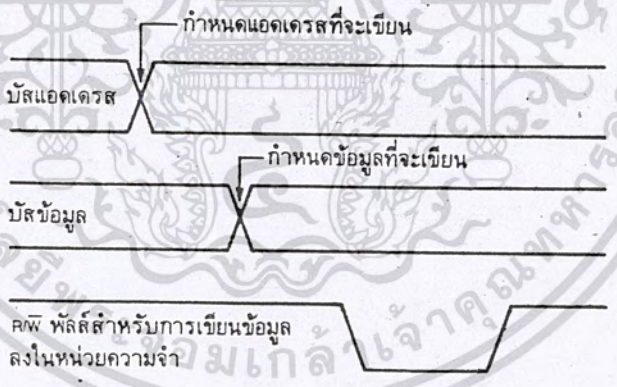
1. ขั้นแรก จะกำหนด แอดเดรสให้กับหน่วยความจำตามตำแหน่งที่ต้องการจะเขียนข้อมูล

2. กำหนดสายสัญญาณข้อมูลเข้าโดยสายสัญญาณ D_{in}

3. ให้รอบรอเวลาชั่วขณะหนึ่ง เรียกช่วงเวลานี้ว่า ช่วงเวลาการเขียน (write access time) เพื่อให้วงจรภายใน ได้รับการถอดรหัส กำหนดตำแหน่งให้เรียบร้อยก่อน และสัญญาณ CE ต้องมารออยู่ก่อนแล้ว

4. หลังจากรอเวลา ให้กำหนดสัญญาณ R/$\overline{W}$ เพื่อการเขียน สัญญาณ R/W นี้จะเป็นพัลส์เล็ก ๆ ที่เพียงพอกับการเขียนข้อมูลลงใน RAM

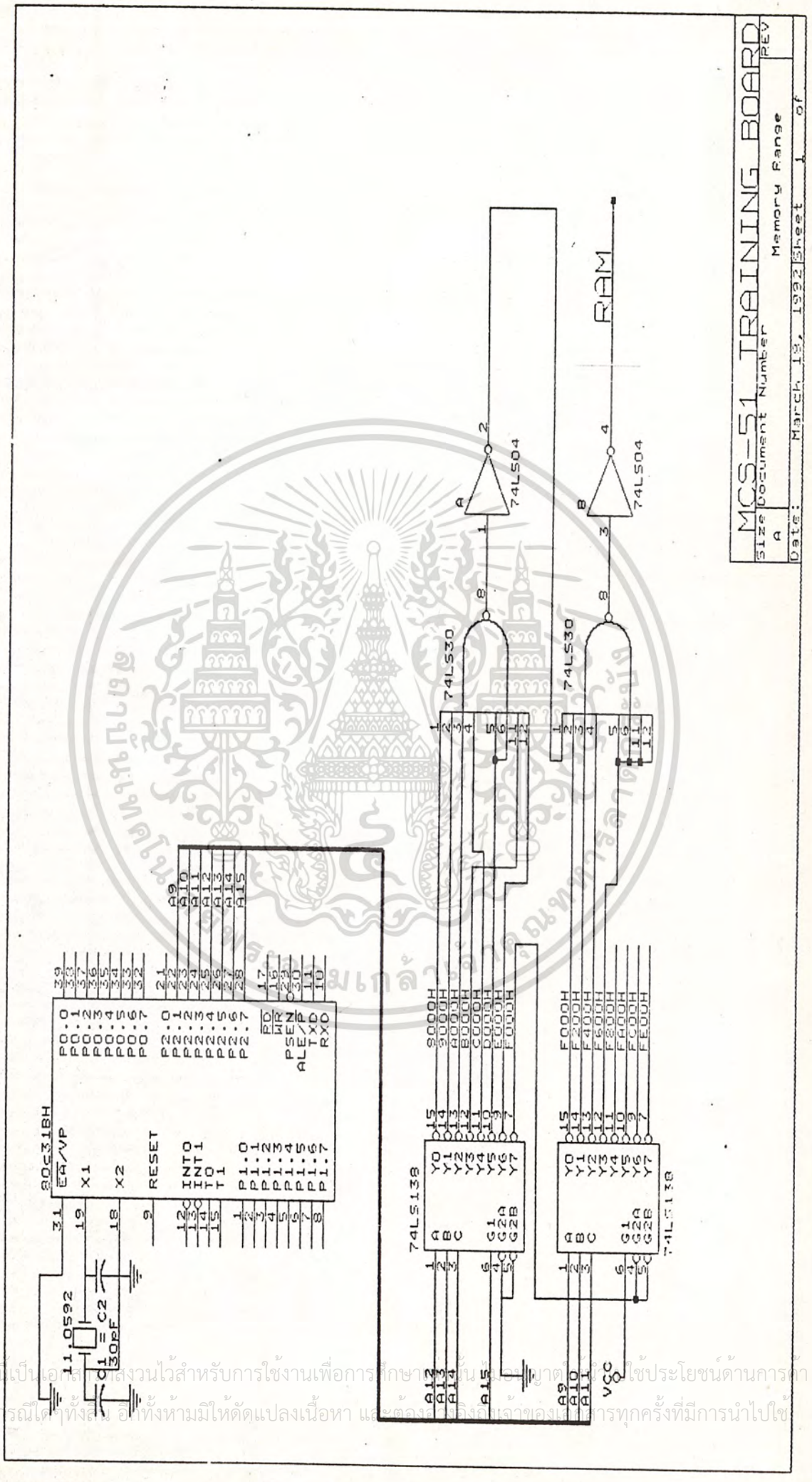
แผนผังเวลาสำหรับการเขียนสามารถแสดงได้ดังรูป



การเชื่อมต่อ RAM เข้ากับ บัส

มีหลักการเช่นเดียวกันกับ EPROM ที่แตกต่างกันจะอยู่ที่ส่วนควบคุมการทำงานของ ไอซี คือ

ขา $\overline{CE}$ (Chip Enable) นำเอาสัญญาณ "RAM" มาต่อเข้าที่ขานี้ โดยสัญญาณ "RAM" ได้มาจากการทำดังรูป



จากวงจรที่แสดงจะเห็นว่าต้องใช้ 74LS138 จำนวน 2 ตัว มาทำการจัดแบ่งช่วงหน่วยความจำ แต่ละตัวมีการทำงานดังนี้

74LS138 ตัวที่ 1

	C	B	A			
A_{15}	A_{14}	A_{13}	A_{12}	$A_{11} - A_0$	แอดเดรส	PIN
1	0	0	0	X	8 X X X	Y_0
1	0	0	1	X	9 X X X	Y_1
1	0	1	0	X	A X X X	Y_2
1	0	1	1	X	B X X X	Y_3
1	1	0	0	X	C X X X	Y_4
1	1	0	1	X	D X X X	Y_5
1	1	1	0	X	E X X X	Y_6
1	1	1	1	X	F X X X	Y_7

74LS138 ตัวที่ 2

	C	B	A				
$A_{15} - A_{12}$	A_{11}	A_{10}	A_9	A_8	$A_7 - A_0$	ช่วงแอดเดรส	PIN
ได้มาจากการดึง	0	0	0	X	X	F0XX-F1XX	Y_0
เอาสัญญาณที่ขา	0	0	1	X	X	F2XX-F3XX	Y_1

Y_7 ของ	0	1	0	X	X	F4XX-F5XX	Y_2
74LS138 ตัวที่	0	1	1	X	X	F6XX-F7XX	Y_3
1 ซึ่งให้สถานะ	1	0	0	X	X	F8XX-F9XX	Y_4
"0" เมื่อเลือก	1	0	1	X	X	FAXX-FBXX	Y_5
ใช้แอดเดรส	1	1	0	X	X	FCXX-FDXX	Y_6
FXXX มาควบ	1	1	1	X	X	FEXX-FFXX	Y_7
คุมการทำงาน							
ของ 74LS138							
ตัวที่ 2							

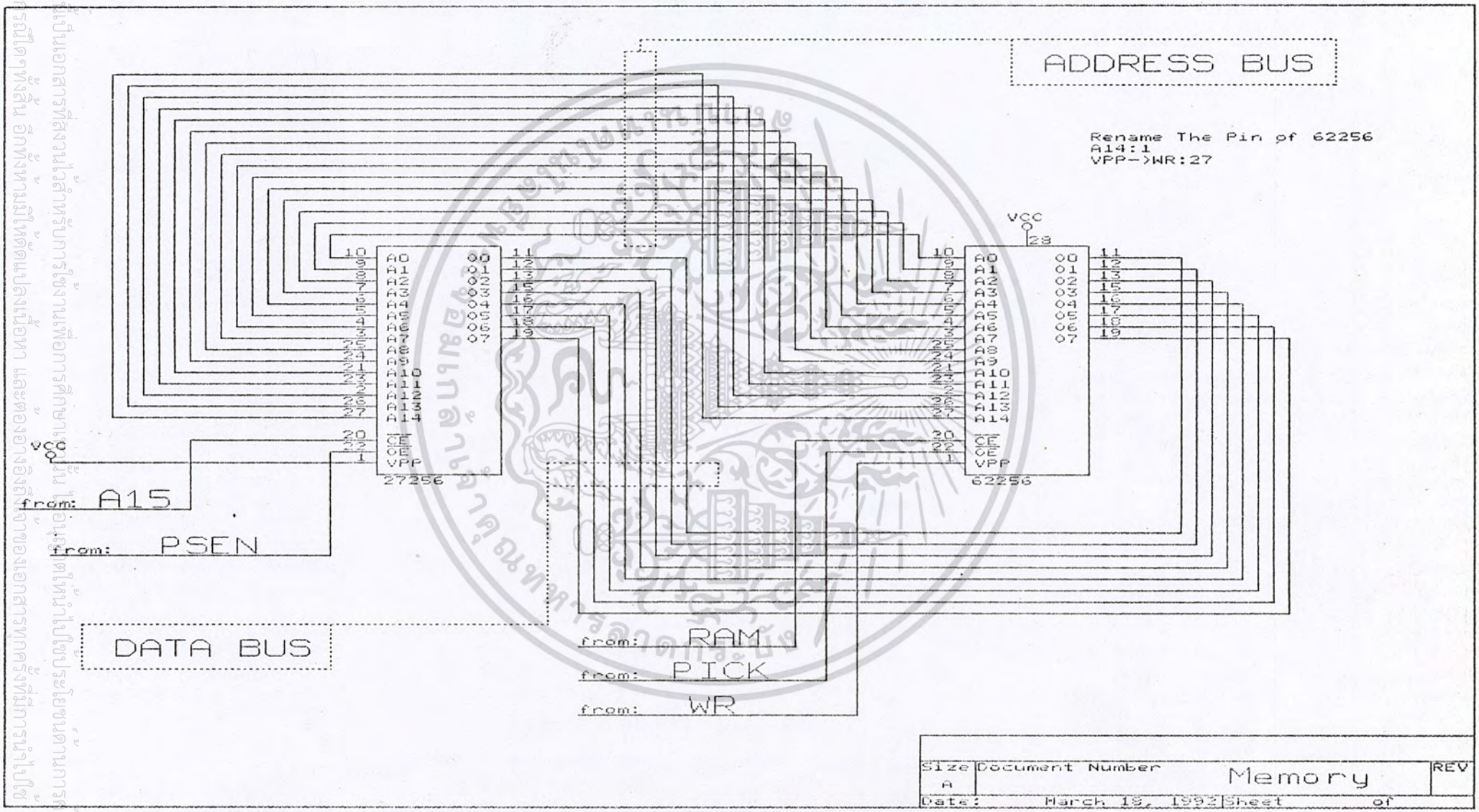
เมื่อได้แอดเดรสต่าง ๆ นี้แล้ว ก็นำมาดำเนินการต่อโดยนำเอาแอดเดรสตั้งแต่ 8XXX - EXXX มาเข้า NAND GATE เบอร์ 74LS30 แล้วต่อออกไปเข้าอินเวอทเตอร์ ออกจากอินเวอทเตอร์ก็นำไปเข้า NAND GATE ตัวที่ 2

และนำเอาแอดเดรส F0XX-F7XX มาต่อเข้า NAND GATE ตัวที่ 2 นี้เช่นกัน แล้วเอาไปผ่านอินเวอทเตอร์อีกทีหนึ่ง ได้สัญญาณที่ออกมาจากอินเวอทเตอร์ให้ชื่อว่า "RAM"

นั่นคือเมื่อมีการเรียกใช้แอดเดรส ตั้งแต่ 8000H-F7FFH จะทำให้ "RAM" มีสถานะเป็น "0"ส่งผลให้ $\overline{CE}$ ของ RAM เกิดการแอ็คทีฟ RAM สามารถทำงานได้

ขา $\overline{OE}$ (Output Enable) นำเอาสัญญาณที่ได้จากการ AND กันของสัญญาณ $\overline{PSEN}$ และ $\overline{RD}$ มาควบคุมการทำงาน นั่นคือ RAM จะให้ข้อมูลออกมาเมื่อ CPU ต้องการติดต่อกับหน่วยความจำภายนอก และอยู่ในลักษณะของการอ่านค่าข้อมูลเข้าไป

ขา $\overline{WR}$ นำเอาสัญญาณ $\overline{WR}$ ของ CPU มาต่อเข้าโดยตรง


ADDRESS BUS
Rename The Pin of 62256
A14:1
VPP->WR:27
VCC
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
CE
OE
VPP
27256
62256
from: A15
from: PSEN
DATA BUS
from: RAM
from: PICK
from: WR
Size A
Document Number
Memory
REV
Date: March 18, 1992
Sheet of

อุปกรณ์อินพุท

ในระบบไมโครโปรเซสเซอร์นั้น ต้องมีการติดต่อกับภายนอกอยู่เสมอหากต้องการนำเอาไมโครโปรเซสเซอร์ไปใช้งานในการควบคุมการทำงานต่างๆ การติดต่อกับภายนอกของ MCS-51 TRAINING BOARD ที่จัดทำขึ้นมานี้ ในส่วนของอินพุทอุปกรณ์หลักก็คือ แผงคีย์บอร์ดจำนวน 49 คีย์ ซึ่งทั้ง 49 คีย์นี้ยังสามารถแบ่งออกเป็น 2 ประเภท คือ คีย์บอร์ดแบบแมทริกซ์

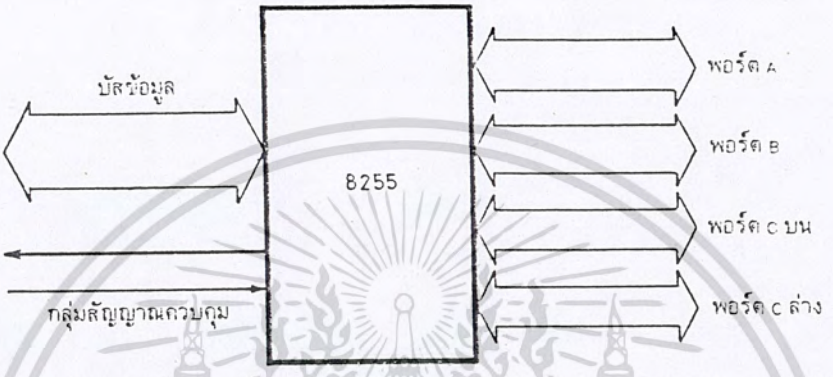
และ คีย์บอร์ดฟังก์ชันพิเศษ

การจัดการกับคีย์บอร์ดทั้ง 2 ประเภทข้างต้น จะทำการติดต่อผ่านทางชิพไอซีเบอร์ 8255 ซึ่งนิยมใช้เป็นพอร์ทข้อมูลแบบขนาน แล้วเขียนโปรแกรมขึ้นมาควบคุมการเชื่อมต่อระหว่าง 8255 และแผงคีย์บอร์ดอีกทีหนึ่ง

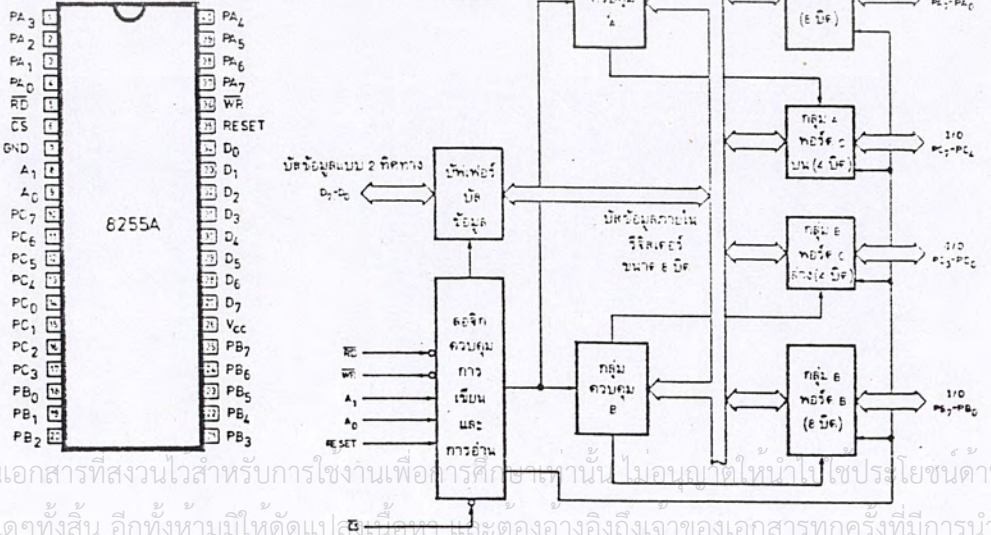
การใช้งานไมโครโปรเซสเซอร์ส่วนใหญ่จะต้องเชื่อมต่อกับอุปกรณ์ภายนอก เช่น สวิทช์ รีเลย์ หรือ ตัวตรวจจับอื่น ๆ การเชื่อมต่อในลักษณะดังกล่าว จะเชื่อมต่อกับพอร์ทอินพุท ในลักษณะที่ง่ายที่สุด คือ การเชื่อมต่อโดยใช้เกตลอจิก 3 สถานะ โดยสัญญาณควบคุมพอร์ทอินพุทจะเป็นตัวไปเปิดเกตให้ข้อมูลเข้าสู่บัส และไมโครโปรเซสเซอร์จะอ่านเข้าไป แต่สำหรับพอร์ทเอาท์พุทจะใช้แลทช์ฟลิปฟลอปทำหน้าที่รับสัญญาณข้อมูล จากไมโครโปรเซสเซอร์ที่ส่งเข้าไปในบัส และได้รับการจับไว้ที่พอร์ท ในขณะที่มีสัญญาณควบคุมพอร์ททริกมาที่ขาแลทช์ พอร์ทอินพุทเอาท์พุทที่ใช้เกตขนาดเล็กดังกล่าว ยังมีจุดอ่อน ในเรื่องของจำนวนไอซี ซึ่งอาจต้องใช้หลายชิป (ถ้าต้องการหลายพอร์ท) และยากที่จะกำหนดลักษณะ การทำงานให้แตกต่างไปจากวงจรเดิมที่ออกแบบไว้ บริษัทผู้ออกแบบไมโครโปรเซสเซอร์ส่วนใหญ่ จึงออกแบบ LSI ชิป เพื่อทำหน้าที่เป็นพอร์ทอินพุตเอาท์พุตของระบบ ซึ่งมีข้อดีในเรื่องการใช้งานได้ง่าย ในบทนี้จะได้กล่าวถึงการประยุกต์ใช้ ICLSI ที่ทำหน้าที่เป็นพอร์ทอินพุตเอาท์พุต ที่รู้จักกันดีมากที่สุด มีราคาถูกและหาได้ง่ายคือ ไอซี8255 ของบริษัทอินเทล

8255 เป็นไอซีในตระกูลของ 8080 ซึ่งบริษัทอินเทลได้ออกแบบมาให้ใช้งานร่วมกับซีพียู 8080 ซึ่งสามารถจะนำมาประยุกต์ใช้กับ MCS-51 ได้เช่นกัน

8255 เป็นไอซีที่มี 40 ขาได้รับการออกแบบมาให้มีสัญญาณ เพื่อเชื่อมต่อกับ 8080 แต่สัญญาณนี้พอเหมาะที่จะใช้กับ MCS-51 ได้เช่นกัน 8255 เป็นไอซีที่ต่อเป็นพอร์ทให้ ไมโครโปรเซสเซอร์ได้ 3 พอร์ท โดยมีโครงสร้างพื้นฐานแสดงได้ดังรูป



การเรียกพอร์ทของ 8255 จะเรียกพอร์ทต่าง ๆ ว่า พอร์ท A พอร์ท B และพอร์ท C โดยพอร์ท C แยกเป็น 2 ส่วนคือ พอร์ท C ล่าง หรือตั้งแต่ PC_0-PC_3 มีจำนวน 4 บิต และพอร์ท C บน หรือตั้งแต่ PC_4-PC_7 ที่พิเศษคือ พอร์ททุกพอร์ทเป็นได้ทั้งพอร์ทอินพุตและพอร์ทเอาท์พุต



เป็นแผนผังภายในของไอซี และการจัดวางขา ของไอซี 8255 การทำงานของวงจร จะใช้สัญญาณควบคุม จากไมโครโปรเซสเซอร์มาควบคุมการทำงานโดยไมโครโปรเซสเซอร์จะส่งคำสั่ง มาโปรแกรมการทำงานหรือกำหนดรูปแบบของพอร์ทให้เป็นอินพุตหรือเอาท์พุตได้

ขาต่าง ๆ ของ 8255

ขาต่าง ๆ ทั้ง 40 ขาของไอซีประกอบด้วย

D_0-D_7 เป็นขาที่ข้อมูลอินพุตเอาท์พุต จะต้องผ่านเข้าออก จากส่วนนี้ D_0-D_7 จึงต่อเข้ากับระบบบัสของไมโครโปรเซสเซอร์ เพื่อให้ไมโครโปรเซสเซอร์สามารถอ่านหรือเขียนข้อมูล จากพอร์ทผ่านทางบัสนี้ได้

$\overline{CS}$ (สัญญาณเลือกชิป) ขานี้เป็นขาอินพุตที่จะรับสัญญาณจากภายนอก เพื่อเลือกชิป 8255 โดยเมื่อขานี้เป็น "0" จะทำให้ 8255 ต่อเข้ากับระบบบัสของไมโครโปรเซสเซอร์ เพื่อให้ไมโครโปรเซสเซอร์เขียนหรืออ่านข้อมูล จากพอร์ทได้

$\overline{RD}$ (สัญญาณการอ่าน) เป็นสัญญาณอินพุตที่ต้องส่งมาจาก ซีพียู เมื่อสัญญาณที่ขานี้เป็น "0" และสัญญาณ CS เป็น "0" ด้วย ไอซี 8255 จะทำตัวให้ซีพียู อ่านข้อมูลจากบัส ในขณะที่เป็นพอร์ทอินพุต

$\overline{WR}$ เป็นสัญญาณการเขียน จะแอ็คทีฟเมื่อสัญญาณ WR และสัญญาณ CS เป็น "0" สัญญาณ นี้จะมาจากซีพียู เมื่อต้องการเขียนข้อมูลลงบนพอร์ทที่กำหนด

A_0 - A_1 (สัญญาณแอดเดรส) ลอจิกของสัญญาณทั้งสอง จะถอดรหัสออกเป็น 4 รหัส เพื่อกำหนดรีจิสเตอร์ภายใน ที่เชื่อมต่อกับพอร์ทอินพุตและเอาท์พุตของ 8255

RESET (สัญญาณรีเซต) เป็นสัญญาณที่ส่งจากภายนอก เข้ามาทำการรีเซต 8255 เพื่อเคลียร์สถานะต่าง ๆ ของ 8255 เมื่อ 8255 ได้รับการรีเซต ก็จะกลับเข้าสู่ โหมดอินพุต หรือทุกพอร์ทที่เป็นพอร์ทอินพุต

PA_0 - PA_7 เป็นสายสัญญาณที่เป็นพอร์ทของ 8255 ที่ชื่อพอร์ท A การเลือกพอร์ทจะเลือกพอร์ทโดยสัญญาณแอดเดรส A_0-A_0

PB_0 - PB_7 เป็นสายสัญญาณที่เป็นพอร์ทของ พอล์ต B ของ 8255 ถูกเลือกโดยสัญญาณ แอดเดรส A_0 -A_1

PC_0 - PC_7 เป็นสายสัญญาณที่เป็นพอร์ท C ของ 8255 การกำหนดพอร์ทนี้จะได้รับการกำหนดโดยสัญญาณแอดเดรส A_0 - A_1 พอร์ท C นี้แบ่งเป็น 2 กลุ่ม คือ กลุ่ม PC_0-PC_3 และกลุ่ม PC_4 -PC_7

รีจิสเตอร์ภายในของ 8255

เมื่อต่อ 8255 แล้ว สิ่งที่ผู้ใช้จะต้องทำก็คือ การโปรแกรมให้ 8255 ทำงานตามที่ต้องการ จากการที่ 8255 มีพอร์ท 4 พอร์ท แต่ละพอร์ทจะเสมือนเป็นรีจิสเตอร์ที่สามารถเขียนและอ่านได้ รีจิสเตอร์แต่ละตัวนี้ จึงถูกกำหนดด้วยแอดเดรสตามที่ตั้งไว้ เช่น ในกรณีที่เป็นแอดเดรส 10H , 11H, 12H และ 13H รีจิสเตอร์แต่ละตัวจะได้รับการกำหนดควบคู่กับสัญญาณ $\overline{RD}$ และ $\overline{WR}$ เพื่อแสดงความหมาย ตัวอย่างเช่น พอร์ท 10H เป็นพอร์ท A ซึ่งเมื่อเขียนที่พอร์ทนี้ จะเป็นการส่งข้อมูลเอาท์พุต และถ้าอ่านพอร์ทนี้ก็จะเป็นการอินพุตข้อมูลจากพอร์ท ดังนั้น สัญญาณของขาควบคุมที่ประกอบกันจะแสดงความหมายดังตาราง

$\overline{RD}$	$\overline{WR}$	$\overline{A1}$	$\overline{A0}$	ความหมาย
1	0	0	0	เขียนพอร์ต A ซึ่งเป็นข้อมูล
0	1	0	0	อ่านพอร์ต A ซึ่งเป็นข้อมูล
1	0	0	1	เขียนพอร์ต B ซึ่งเป็นข้อมูล
0	1	0	1	อ่านพอร์ต B ซึ่งเป็นข้อมูล
1	0	1	0	เขียนพอร์ต C ซึ่งเป็นข้อมูล
0	1	1	0	อ่านพอร์ต C ซึ่งเป็นข้อมูล
1	0	1	1	เขียนข้อมูล ซึ่งเป็นรหัสควบคุม
0	1	1	1	อ่านเข้ามา ซึ่งไม่มีความหมายใด

การใช้งาน 8255 จะต้องส่งรหัสควบคุม (control code) เข้าไปยังพอร์ทข้อมูลควบคุมเพื่อควบคุมการทำงานของ 8255 โดยใช้สัญญาณควบคุมพอร์ทหมายเลข 13H การควบคุมการทำงานของ 8255 มีหลายโหมด แต่ละโหมดจะแตกต่างกันออกไป การโปรแกรมให้ 8255 ทำงานจะทำได้ 3 โหมดคือ โหมด 0 โหมด 1 และโหมด 2 ซึ่งบนบอร์ดนี้จะโปรแกรมให้ 8255 ทำงานในโหมด 0

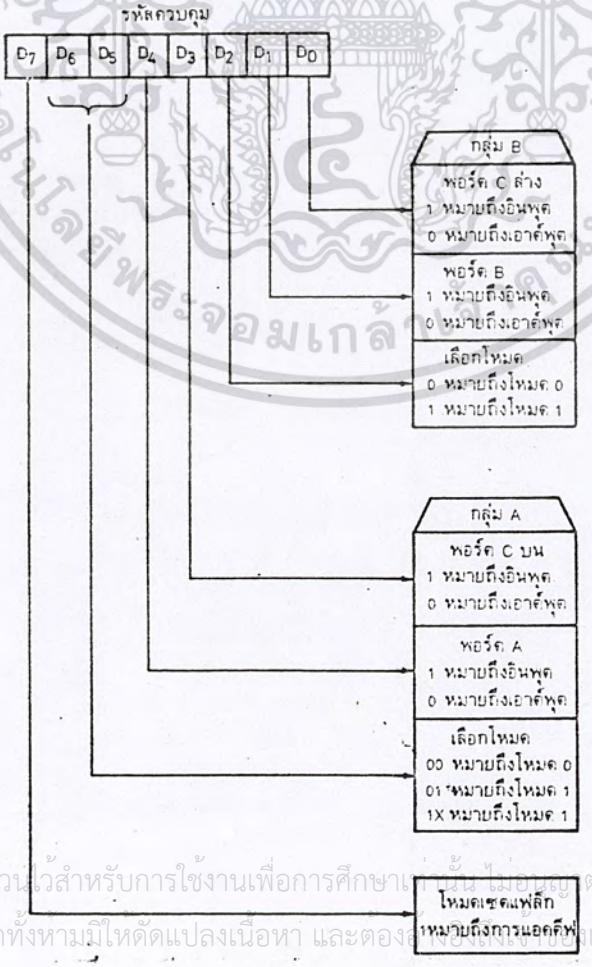
<u>โหมด 0 หรืออินพุตเอาท์พุตแบบพื้นฐาน</u>

การกำหนดโหมดการทำงาน จะต้องส่งข้อมูลคำสั่งเข้าไปโปรแกรมในพอร์ทควบคุมของ 8255 แต่ละบิตของข้อมูลที่ส่งไปจะมีความหมายในตัวเอง

การโปรแกรม 8255 คือ การให้ค่ารหัสบิตต่าง ๆ เข้าไปในรหัสควบคุมแล้วส่งไปยังรีจิสเตอร์ ของพอร์ทควบคุม ความหมายของบิตต่าง ๆ มีดังนี้

บิต D_7 เป็นบิตที่แสดงรหัสคำสั่งควบคุม ถ้าบิตนี้เป็น "1" หมายถึงรหัสควบคุมนี้จะมีผลต่อ การเปลี่ยนแปลงการเซตโหมดต่าง ๆ ของ 8255

บิต D_6 และ D_5 เป็นการเลือกโหมดของพอร์ท A ซึ่งมี 3 โหมดคือ โหมด 0 โหมด 1 และ โหมด 2 ดังแสดงในรูป



บิต D_3 เป็นบิตที่บอกถึงการเซตของพอร์ท C บน ถ้าเป็น "0" จะทำให้พอร์ท C บน เป็นเอาท์พุต

บิต D_2 เป็นบิตที่บอกถึงการเซตโหมดของพอร์ท B ถ้าเป็น "0" หมายถึง การเลือกพอร์ท B เป็นโหมด 0 และถ้าเป็น "1" หมายถึงการเลือกโหมด 1

บิต D_1 เป็นการกำหนดอินพุตเอาท์พุตของพอร์ท B ถ้าเป็น "0" หมายถึง เอาท์พุต ถ้าเป็น "1" หมายถึงอินพุต

บิต D_0 เป็นการกำหนดอินพุตเอาท์พุตของพอร์ท C ล่าง ถ้าเป็น "0" หมายถึงเอาท์พุต ถ้าเป็น "1" หมายถึง อินพุต

การโปรแกรม 8255 จะเริ่มจากการเซตค่าที่ต้องการแล้วเอาท์พุตไปยังพอร์ทควบคุม เช่น ถ้าต้องการโปรแกรมให้ทั้ง พอร์ท A,B และ C เป็นพอร์ทเอาท์พุตหมด เราจะเลือก 8255 ให้อยู่ในโหมด 0 โดยมีรหัสควบคุมเป็น 10000000 หรือ 80H ดังนั้นจึงเขียนคำสั่งได้เป็น

```
LD A, 80H    หมายถึงกำหนดรหัสควบคุม
MOVX (13H), A  หมายถึงส่งไปยังพอร์ทควบคุม
```

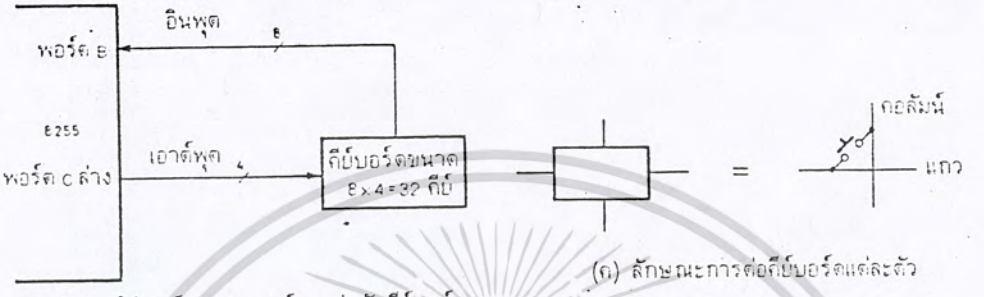
หลังจากที่กระทำคำสั่งผ่านไปแล้ว พอร์ท A,B และ C จะเป็นพอร์ทเอาท์พุตหมด ซึ่งก็จะส่งข้อมูลจากซีพียูไปยังพอร์ทต่าง ๆ ได้ เช่น ถ้าต้องการส่งข้อมูล 8AH ไปยังพอร์ท A ข้อมูล 41H ไปยัง พอร์ท B และข้อมูล 25H ไปยังพอร์ท C คำสั่งที่ใช้คือ

```
LD A, 8AH      หมายถึงเลือกค่า 8AH
MOVX (10H), A  หมายถึงส่งให้พอร์ท A
LD A, 41H      หมายถึงเลือกค่า 41H
MOVX (11H), A  หมายถึงส่งให้พอร์ท B
LD A, 25H      หมายถึงเลือกค่า 25H
MOVX (12H), A  หมายถึงส่งให้พอร์ท C
```

เนื่องจากมีพอร์ทที่รับส่งข้อมูล 3 พอร์ท คือ พอร์ท A พอร์ท B และพอร์ทCซึ่งพอร์ท C จะแยกออกเป็น 2 ส่วน คือ พอร์ท C ล่าง และพอร์ท C บน

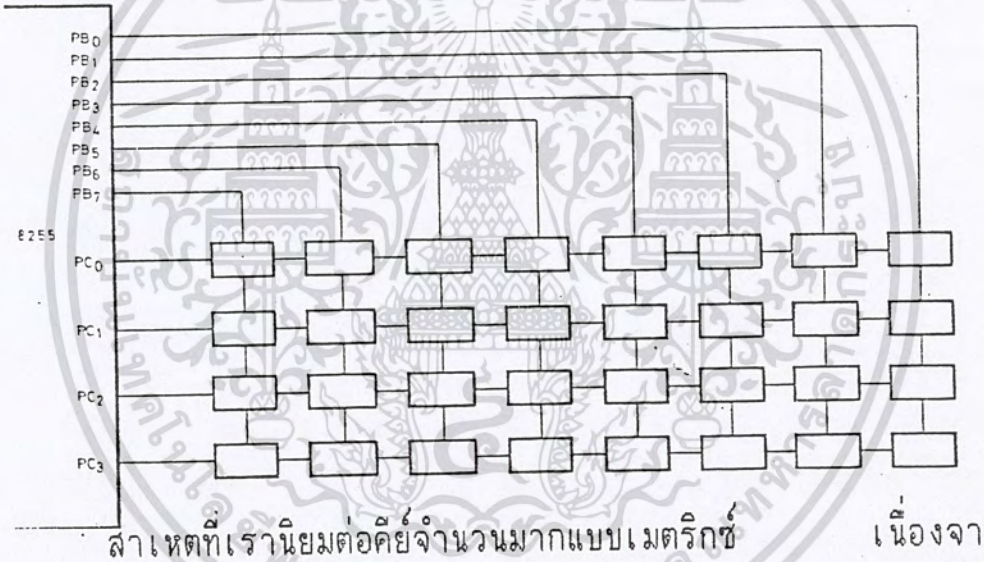
เราสามารถโปรแกรมให้ทั้ง 4 พอร์ทนี้เป็นอินพุตหรือเอาท์พุตก็ได้ เช่น ถ้าให้รหัสควบคุมเป็น 82H จะทำให้พอร์ท B เป็นอินพุต พอร์ท A และพอร์ท C เป็นเอาท์พุต

ตัวอย่างการใช้งานของ 8255 ในโหมด 0 นี้ เช่น เมื่อต้องการให้พอร์ท B เป็นอินพุตและพอร์ท C ล่างเป็นเอาท์พุต เพื่อรับรู้การกดคีย์บอร์ด และหาดูว่ามีคีย์ใดกด ซึ่งเราสามารถใช้การจัดคีย์บอร์ดในรูปแบบเมตริกซ์ได้ดังรูป



(ก) ลักษณะการต่อคีย์บอร์ดแต่ละตัว

(ก) กำหนดให้พอร์ต B และพอร์ต C ต่อกับคีย์บอร์ด



สาเหตุที่เรานิยมต่อคีย์จำนวนมากแบบเมตริกซ์ เนื่องจากเป็นโครงสร้างทางฮาร์ดแวร์ แบบประหยัด และช่วยลดข้อยุ่งยากต่าง ๆ ได้ แต่ก็ยังต้องใช้ซอฟต์แวร์ในการควบคุมหรือตรวจสอบว่า คีย์ใดกด ในรูป เป็นการต่อแบบเมตริกซ์ ซึ่งมีจำนวนแถว 4 แถว และคอลัมน์ 8 คอลัมน์ ทำให้ได้จำนวนคีย์ทั้งสิ้น 32 คีย์

หลักการทำงานทั่วไป เราจะทำการสแกนดู กล่าวคือ กำหนดให้แต่ละแถวซึ่งเป็นพอร์ทเอาท์พุตเป็น "0" หรือ "1" ในเวลาต่างกัน เช่น แถวแรกเป็น "0" แถวอื่นเป็น "1" หมด แล้วทำการอ่านข้อมูลที่พอร์ทอินพุตดูว่า มีบิตใดบิตหนึ่งทางคอลัมน์เป็น "0" หรือไม่ ถ้ามีก็ทราบได้ว่าคีย์ใน ตำแหน่งแถวแรกและคอลัมน์ที่

เท่าไรเป็นคีย์ที่ได้รับการกด แต่ถ้าไม่มี ซีพียู ก็จะสแกนไปยังแถวถัดไป และวนรอบไปเรื่อย ๆ ตลอดเวลา ดังนั้น การตรวจสอบคีย์กดจะทำให้ทราบว่า แถวหรือคอลัมน์ที่เท่าไร ซึ่งเป็นคีย์ที่ได้รับการกด

การอินเตอร์เฟสระหว่าง 8255 และ 80C31

องค์ประกอบทางด้านฮาร์ดแวร์ที่จัดสร้างขึ้นมานี้ นำเอาชิปไอซี 8255 มาใช้งานจำนวน 2 ตัวด้วยกัน แต่ละตัวจะทำหน้าที่แตกต่างกันดังจะกล่าวต่อไปนี้คือ

ชิปไอซี 8255 ตัวที่ 1

ใช้ในส่วนของระบบอินพุท เอาท์พุต โดยอินพุตจะอยู่ในเรื่องของการสแกนคีย์บอร์ด และเอาท์พุตจะทำการแสดงผลออกทางจอ LCD (Liquified Crystal Diode) โดย

$PC_7 - PC_0$

$PB_7 - PB_4$ ใช้ในการสแกนคีย์บอร์ด

$PA_7 - PA_0$

$PB_2 - PB_0$ ใช้ในการแสดงผลออกทาง LCD

การจัดการเกี่ยวกับคีย์บอร์ด

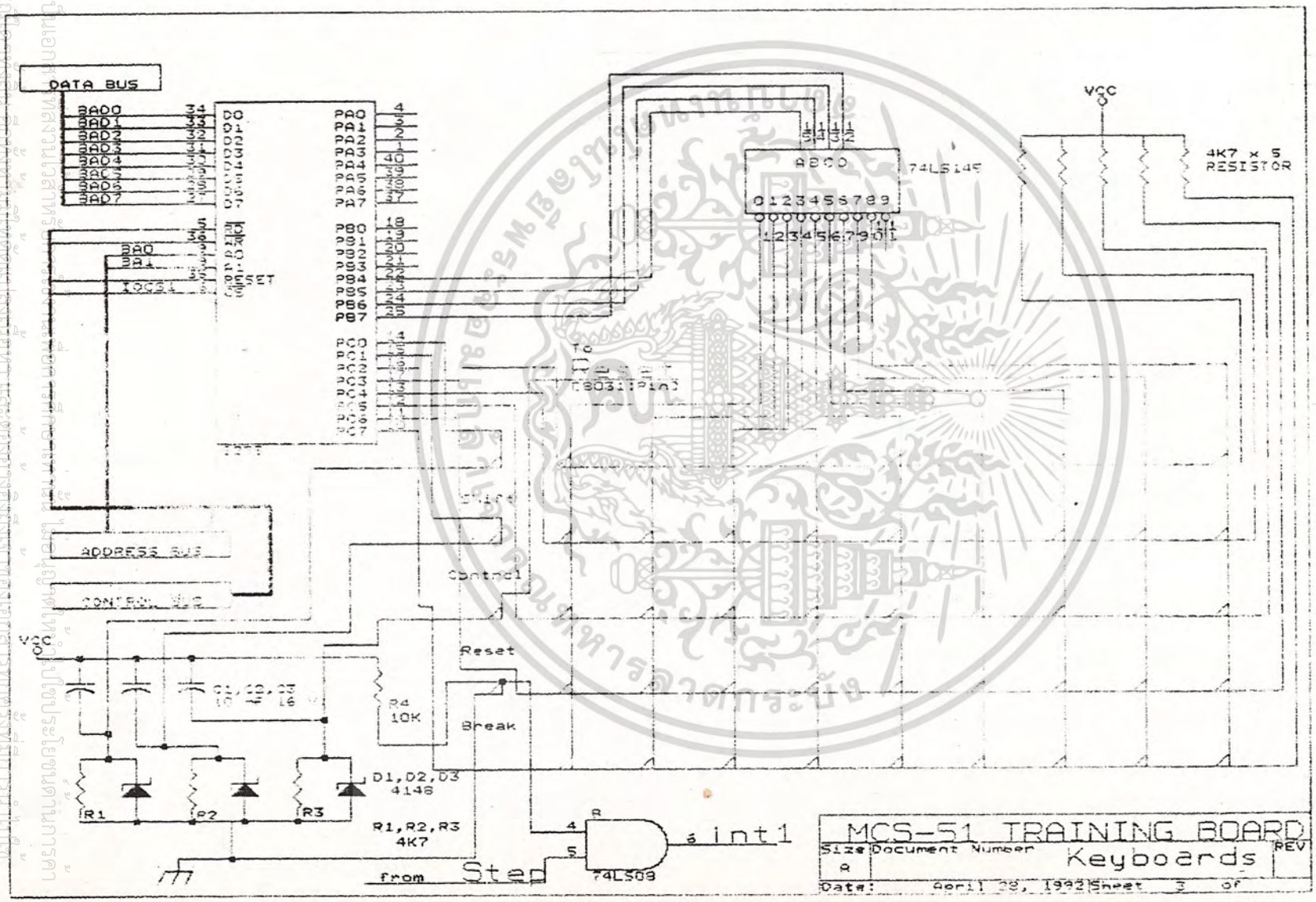
ในส่วนของคีย์บอร์ดนี้ เราใช้อุปกรณ์เหล่านี้ คือ

- ชิปไอซี เบอร์ 8255
- สวิทช์สำหรับทำคีย์บอร์ดแบบ Dot Matrix
- ชิปไปซี 74LS 145
- อื่น ๆ

ซึ่งอุปกรณ์แต่ละตัวทำหน้าที่ต่อไปนี้

8255 พอร์ทข้อมูลแบบขนาน

ลักษณะการทำงานทั่ว ๆ ไป ก็เช่นเดียวกับหลักการของการแสดงผลนั่นเอง แต่ที่แตกต่างกันก็อยู่ที่สายสัญญาณบางตัวที่ใช้ ในการควบคุมการทำงานของไอซี และขาสัญญาณที่ส่งออกไปควบคุมอุปกรณ์อื่น ตามพอร์ทต่าง ๆ ทางด้านการควบคุมการทำงาน ของไอซีนั้น มีดังนี้


DATA BUS
BAD0
BAD1
BAD2
BAD3
BAD4
BAD5
BAD6
BAD7
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
RESET
ADDRESS BUS
CONTROL BUS
Control
Reset
Break
R4
10K
R1
R2
R3
D1,D2,D3
4148
R1,R2,R3
4K7
from Step
74LS08
int1
74LS145
4K7 x 5
RESISTOR
VCC
MCS-51 TRAINING BOARD
Keyboards
Size
A
Document Number
REV
Date: April 28, 1992
Sheet 3 of

ข้อมูลที่ต้องการใช้จริง ๆ จะอยู่ที่ PC_7 -PC_3 เท่านั้น ส่วนที่ PC_2- PC_0 จะมีค่าข้อมูลเป็นอะไรก็ได้

ดังนั้นกำหนดให้ว่าเมื่อไม่มีการกดคีย์ใด ๆ ค่าข้อมูลที่อ่านได้จากพอร์ท C ให้มีค่า FFH

ด้วยหลักการนี้ เมื่อใดก็ตามที่อ่านค่าเข้ามาแล้ว ค่าที่ได้ไม่เท่ากับค่านี้ แสดงว่ามีการกดคีย์เกิดขึ้น

แผงคีย์บอร์ดสามารถแบ่งออกได้เป็น 2 ประเภท คือ

คีย์บอร์ดธรรมดา

ประกอบอยู่ในรูปของเมทริกซ์ขนาด 9x5 เป็นคีย์บอร์ดที่ใช้ในการป้อนตัวอักษรภาษาอังกฤษตั้งแต่ A-Z ตัวเลขตั้งแต่ 0-9 และตัวอักษรพิเศษที่มีใช้ในภาษาแอสเซมบลี้ของซีพียูตระกูล MCS-51 โดยเฉพาะ แผนภาพวงจรแสดงได้ดังหน้าถัดไป

คีย์บอร์ดฟังก์ชันพิเศษ

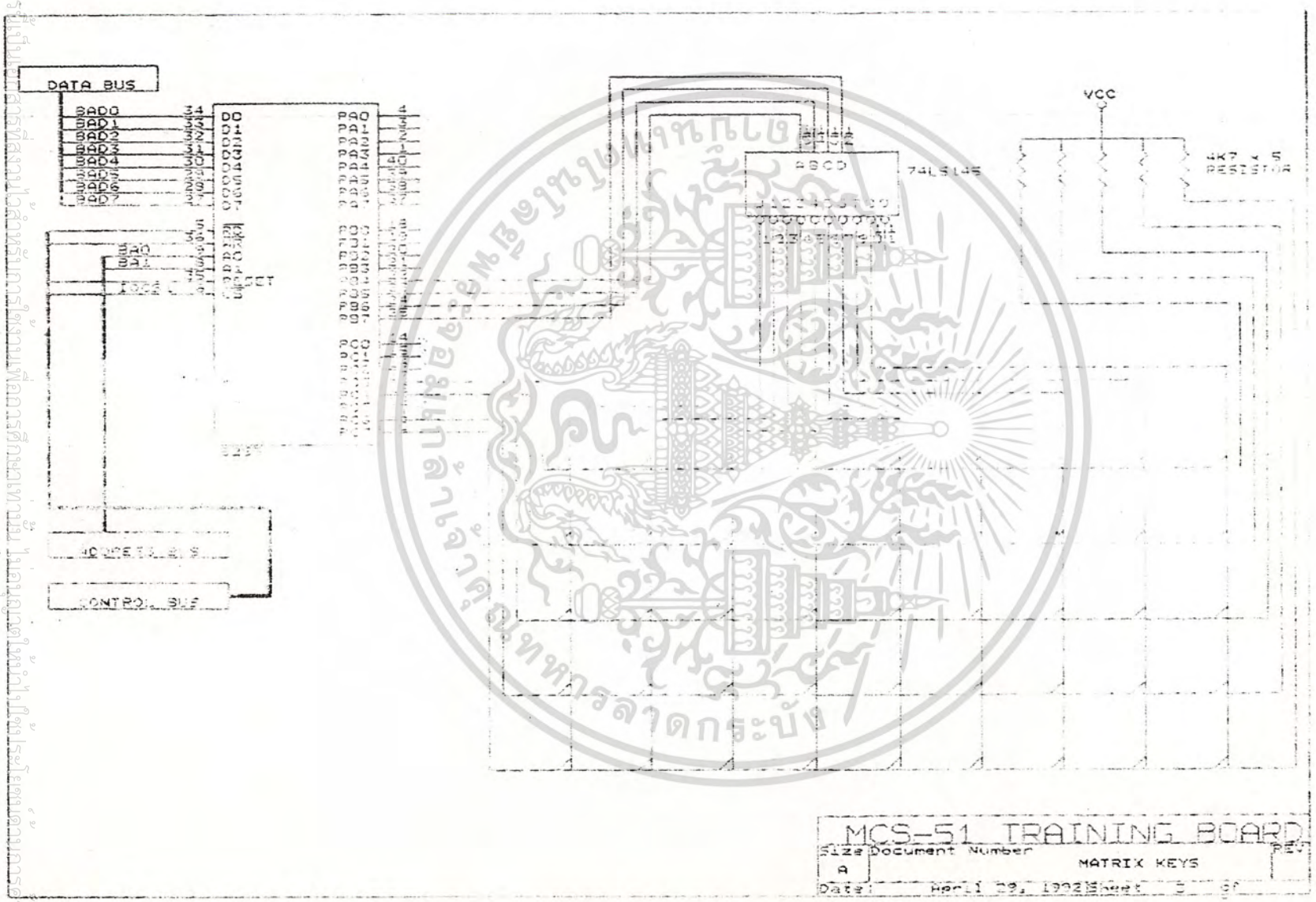
มีอยู่ด้วยกันจำนวน 4 คีย์ คือ

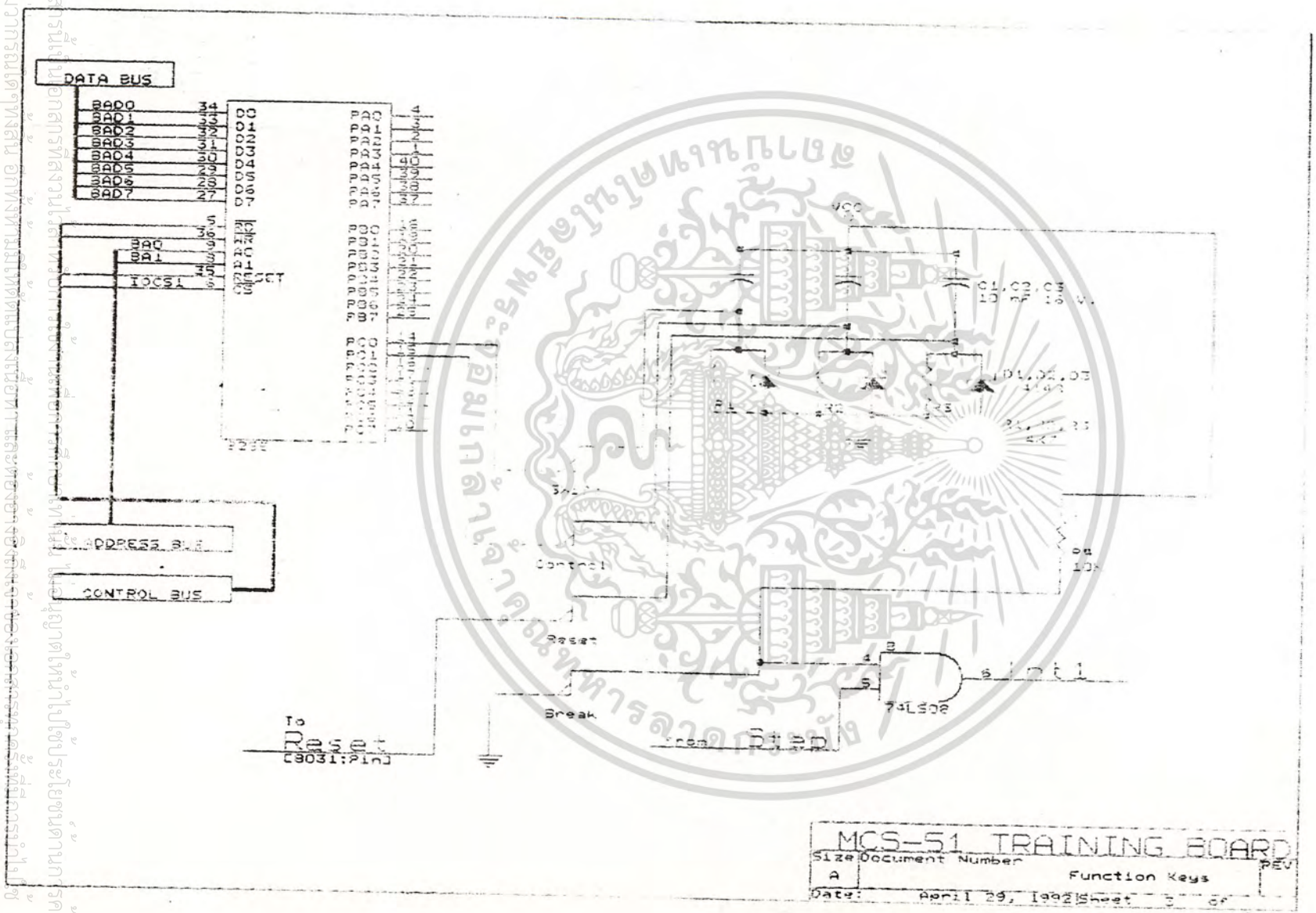
SHIFT, CONTROL > เมื่อไม่มีการกดคีย์ อยู่ในสถานะ "0"
เมื่อมีการกดคีย์ อยู่ในสถานะ "1"

RESET

BREAK

คีย์ Shift และคีย์ Control ใช้ในการป้อนโปรแกรมและในช่วงของการรันทดสอบ ส่วนคีย์ Reset และคีย์ Break ใช้ในการควบคุมการทำงานของระบบ ลักษณะโครงสร้างของคีย์ต่างๆเหล่านี้ สามารถแสดงได้ดังแผนภาพวงจรถัดจากแผนภาพวงจรของคีย์บอร์ดธรรมดา


DATA BUS
BAD0
BAD1
BAD2
BAD3
BAD4
BAD5
BAD6
BAD7
74LS145
VCC
4K7 x 5
RESISTOR
MCS-51 TRAINING BOARD
MATRIX KEYS
Size
A
Document Number
Rev
Date: April 29, 1992 Sheet 3 of


DATA BUS
BAD0
BAD1
BAD2
BAD3
BAD4
BAD5
BAD6
BAD7
BA0
BA1
IOCS1
ADDRESS BUS
CONTROL BUS
Reset
Break
To
Reset
C8031:Pin3
VCC
74LS08
MCS-51 TRAINING BOARD
Function Keys
Size
A
Document Number
REV
Date:
April 29, 1992
Sheet
of

อุปกรณ์แสดงผล

ในส่วนของอุปกรณ์แสดงผลนี้ นำเอาแอลซีดีมาใช้งาน ทั้งนี้ก็เพื่อความชัดเจนในการอ่านตัวอักษรและสัญลักษณ์ต่างๆนั่นเอง

องค์ประกอบใหญ่ๆแบ่งได้เป็น

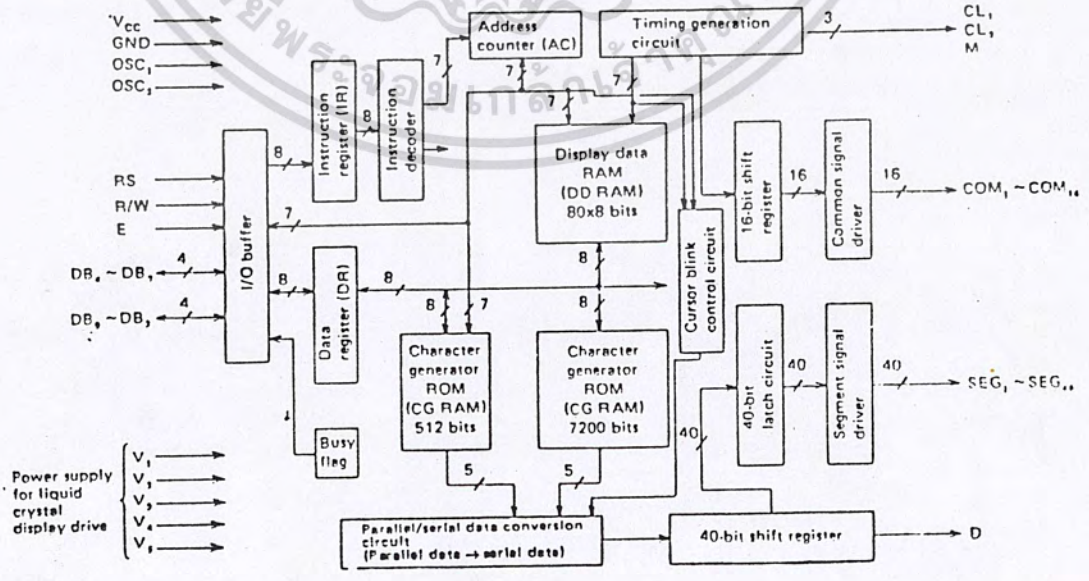
1.DOT MATRIX LCD เป็นตัวแสดงผลให้สามารถมองเห็นในลักษณะการปิดและเปิดตัวเองกับแสง หรือส่วนที่เป็นตัวกระจกบรรจุผลึกนั่นเอง

2.DRIVER เป็นตัวรับสัญญาณจากตัวควบคุมมาขับผลึก LCD อีกทีหนึ่ง เบอร์ที่นิยมใช้ใน LCD MODULE ก็คือ HD44100H,MSM5259

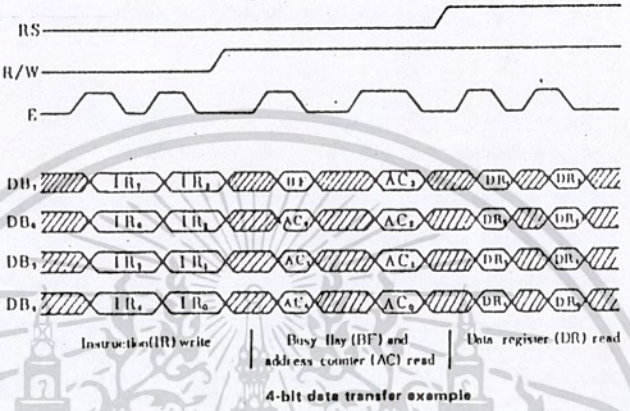
3.CONTROLLER เป็นตัวรับข้อมูลจากอุปกรณ์ภายนอกมา และจัดการควบคุม LCD MODULE ให้ทำงานแสดงผลต่างๆ เช่น การลบจอภาพ การเกิดตัวอักษร โดยมีไอซีเบอร์ที่นิยใช้ คือ HD61830

โดยมาก LCD MODULE ในแต่ละบริษัทจะใช้ตัว CONTROLLER ที่มีหลักการทำงานเหมือนๆกัน HD44780จะควบคุม LCD ให้แสดงผลในรูปตัวอักษรหรือสัญลักษณ์ต่างๆ สามารถต่อใช้งานแบบ 4 บิท หรือ 8 บิท ก็ได้

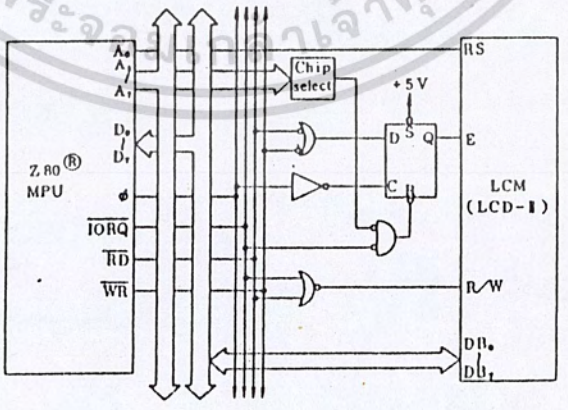
Block diagram of HD44780 interior



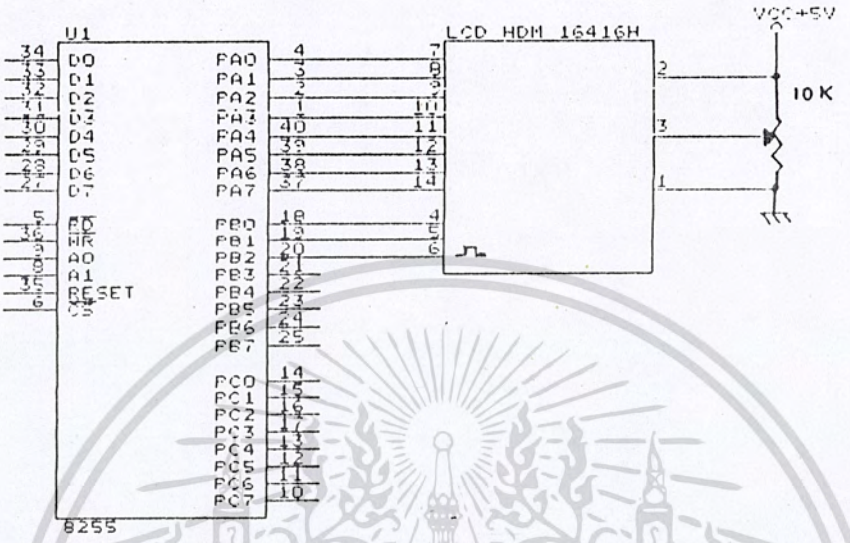
ข้อมูลที่ส่งนั้น ครั้งแรกจะถือว่าเป็นข้อมูล 4 บิทบน
ครั้งที่สองจะถือว่าเป็นข้อมูล 4 บิทล่าง
ดัง Timing Diagram ข้างล่างนี้



ลักษณะการต่อใช้งานจะเป็นดังรูปข้างล่างนี้



การต่อใช้งาน



จากวงจรเป็นการต่อ LCD เข้ากับ 8255 โดยมีสัญญาณต่างๆดังนี้

Port A _____ Data Port

[PA_7-PA_0]

Port B _____ Control Signal

[PB_3-PB_0]

ขาต่างๆที่ต่อใช้งาน

1. RS (Register Selection) เป็นขาที่ใช้ในการเลือกรีจิสเตอร์ภายใน ซึ่งมีอยู่ด้วยกัน 2 ตัว คือ Instruction Register(IR)

และ Data Register(DR)

โดยถ้าเป็น "1" เป็นการเลือก Data

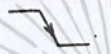
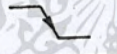
"0" เป็นการเลือก Instruction

2.R/W (Read/Write) เลือกว่าจะเขียนหรืออ่านข้อมูล

โดยถ้าเป็น "1" เป็นการอ่านข้อมูล

"0" เป็นการเขียนข้อมูล

The relation between the operation and the combination of RS, R/W

RS	RW	E	OPERATION
0	0		Write instruction code
0	1		Read busy flag and address counter
1	0		Write data
1	1		Read data

When performing data and instruction code by 4 bit, transfer RS, R/W every time.

3.E (Enable Signal) ขาที่ใช้รับสภาพว่าจะทำการเขียนหรืออ่านข้อมูล

4.DB_7-DB_0 ใช้รับส่งข้อมูล

5.V_{DD} ไฟเลี้ยงวงจร

6.V_{SS} ขากราวด์

7.V_0 ขารับแรงดันใช้ในการขับ LCD ให้สว่างหรือมืด

เมื่อเริ่มป้อนไฟให้แก่วงจร HD44780จะทำการรีเซ็ทตัวเองโดยใช้เวลาประมาณ 10 mS. หลังจากไฟ V_{DD} ขึ้นถึง 4.5 volt แล้ว มีขั้นตอนดังนี้

1.Display Clear ทำการลบข้อมูลจอภาพ LCD

2.Function Set ทำการเซ็ทค่าภายใน

DL=1 : ทำการเซ็ทให้ติดต่อแบบ 8 บิท

N =0 : ทำการเซ็ทให้แสดงผล 1 บรรทัด

F =0 : ทำการเซ็ทให้แสดงผลแบบ 5x7 Dot ต่อตัวอักษร

3.Display On/Off โดย D=0 : Display Off

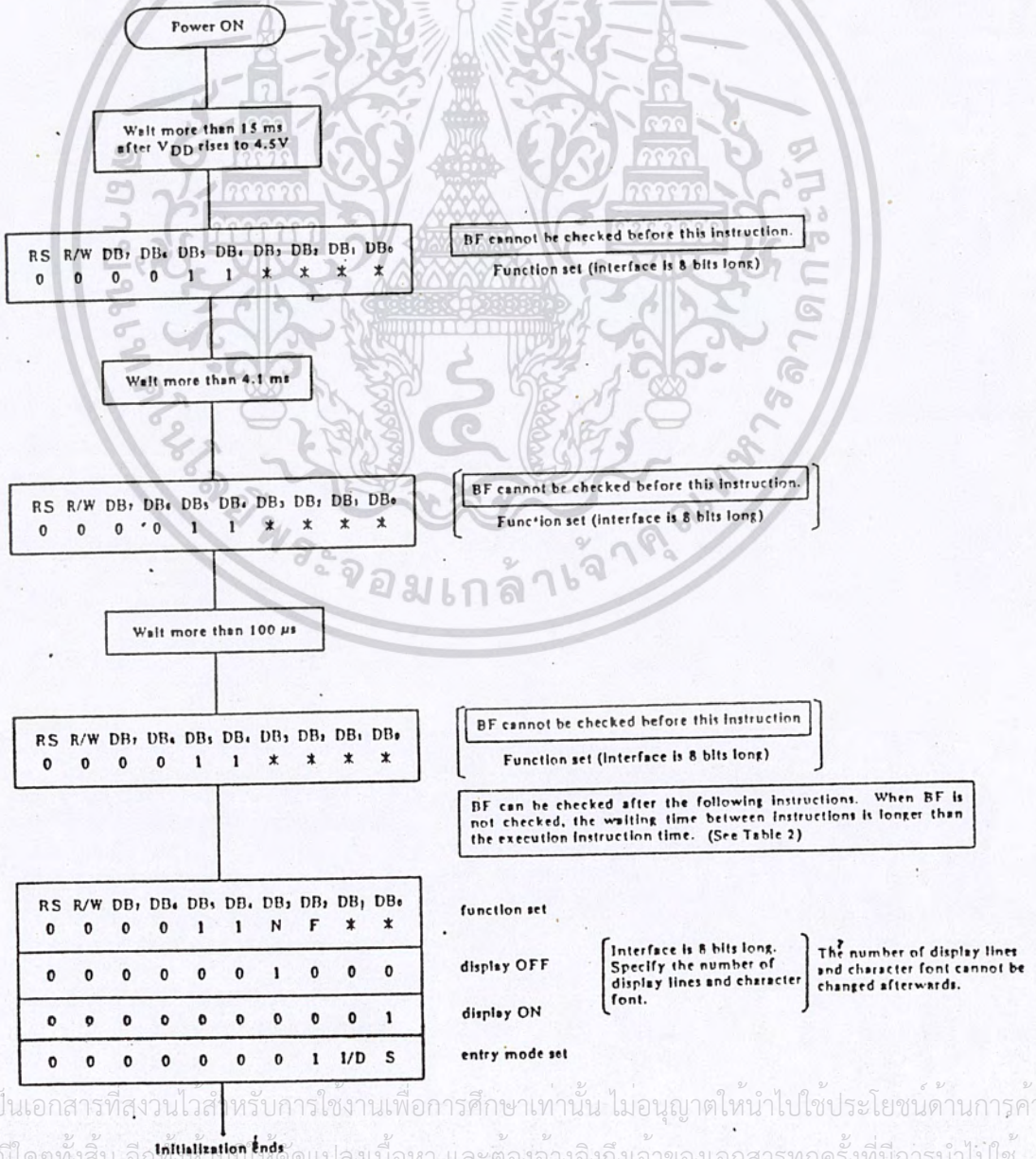
C=0 : Cursor Off

B=0 : Blink Off

4.Entry Mode Set โดย I/D=1 : Increse Counter

S =0 : No Shift

สามารถแสดงได้ดังโฟลชาร์ทนี้



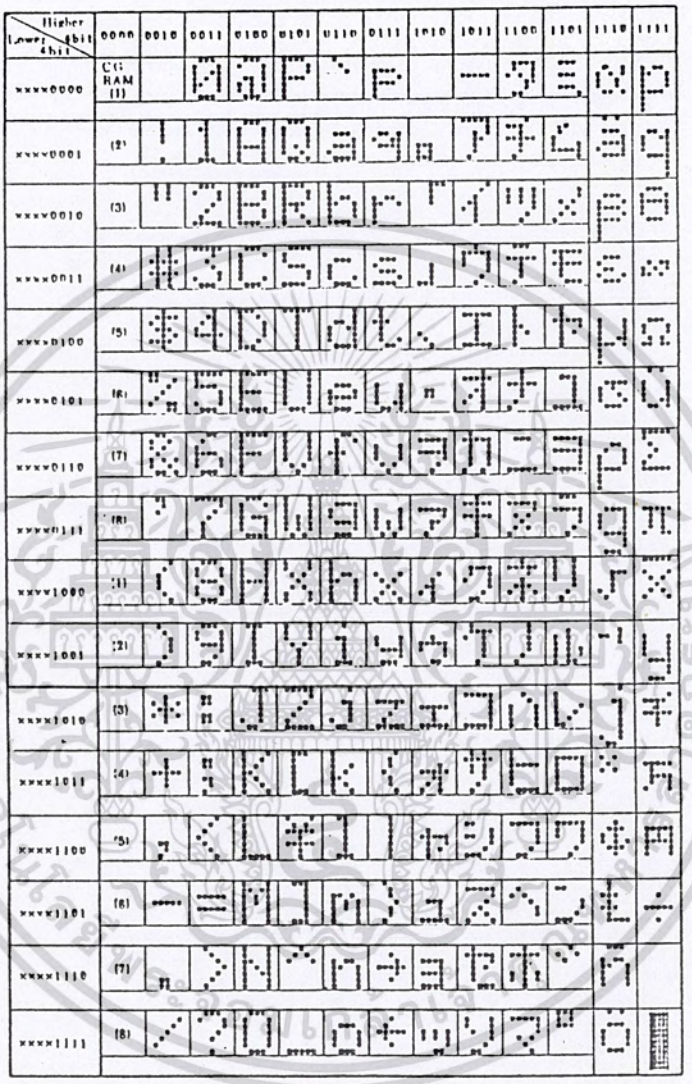
INSTRUCTION TABLE

Instruction	Code										Description	Execution time (when fosc is 250 kHz) Note 1	Execution time (when fosc is 160 kHz) Note 2
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0			
Clear display	0	0	0	0	0	0	0	0	0	1	Clears all display and returns the cursor to the home position (Address 0).	82 μs ~ 1.64 ms	120 μs ~ 4.9 ms
Return home	0	0	0	0	0	0	0	0	1	*	Returns the cursor to the home position (Address 0). Also returns the display being shifted to the original position. DD RAM contents remain unchanged.	40 μs ~ 1.6 ms	120 μs ~ 4.8 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets the cursor move direction and specifies or not to shift the display. These operations are performed during data write and read.	40 μs	120 μs
Display ON/ OFF control	0	0	0	0	0	0	1	D	C	B	Sets ON/OFF of all display (D), cursor ON/OFF (C), and blink of cursor position character (B).	40 μs	120 μs
Cursor and display shift	0	0	0	0	0	1	S/C	R/L	*	*	Moves the cursor and shifts the display without changing DD RAM contents	40 μs	120 μs
Function set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL) number of display lines (L) and character font (F).	40 μs	120 μs
Set CG RAM address.	0	0	0	1	A_{CG}						Sets the CG RAM address. CG RAM data is sent and received after this setting.	40 μs	120 μs
Set DD RAM address	0	0	1	A_{DD}							Sets the DD RAM address. DD RAM data is sent and received after this setting.	40 μs	120 μs
Read busy flag & address	0	1	BF	AC							Reads Busy flag (BF) indicating internal operation is being performed and reads address counter contents.	1μs	1 μs
Write data to CG or DD RAM	1	0	Write Data								Writes data into DD RAM or CG RAM.	40 μs	120 μs
Read data to CG or DD RAM	1	1	Read Data								Reads data from DD RAM or CG RAM.	40 μs	120 μs
	I/D = 1: Increment (+1) I/D = 0: Decrement (−1) S = 1: Accompanies display shift. S/C = 1: Display shift S/C = 0: Cursor move R/L = 1: Shift to the right. R/L = 0: Shift to the left. DL = 1: 8 bits DL = 0: 4 bits N = 1: 2 lines N = 0: 1 line F = 1: 5 x 10 dots F = 0: 5 x 7 dots BF = 1: Internally operating BF = 0: Can accept instruction										DD RAM: Display data RAM CG RAM: Character generator RAM A_{CG}: CG RAM address A_{DD}: DD RAM address Corresponds to cursor address. AC: Address counter used for both of DD and CG RAM address.	Execution time changes when frequency changes. (Example) When fosc is 270 kHz: $40\ \mu s \times \frac{250}{270} = 37\ \mu s$	

*No effect

Notes 1. Applied to models driven by 1/8 duty or 1/11 duty.
2. Applied to models driven by 1/16 duty.

CHARACTER FONT TABLE

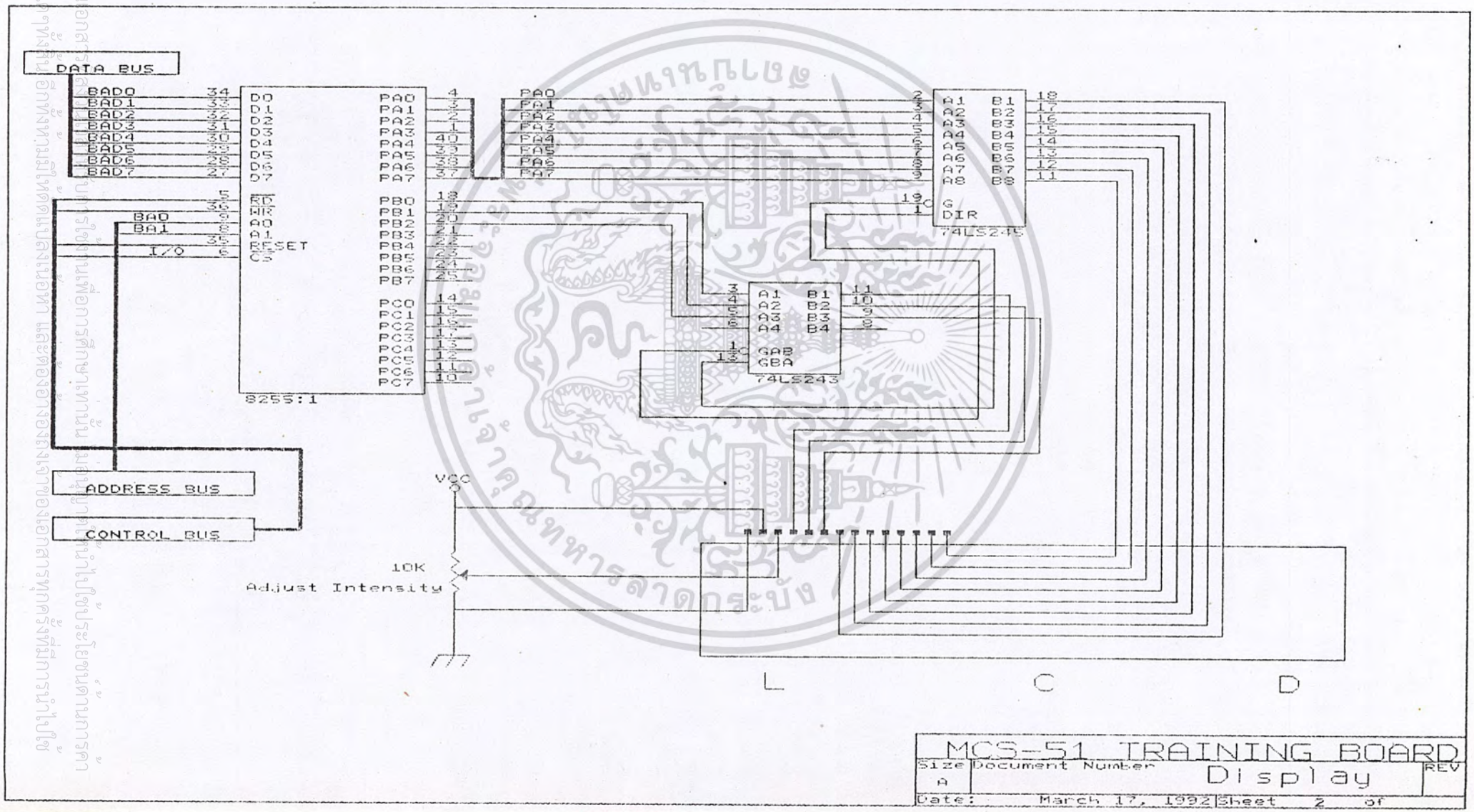


NOTE: CGRAM is a CHARACTER GENERATOR RAM having a storage function of character pattern which enable to change freely by user's program.

การสร้างรูปแบบตัวอักษรขึ้นเอง

จากตารางตัวอักษร จะเห็นว่าตำแหน่งในตาราง 00H-07H ที่สามารถเขียนข้อมูลกำหนดเองได้

การเขียนข้อมูลต้องทำการกำหนดแอดเดรสของ CG RAM ก่อน โดยเขียนได้ 64 ตำแหน่ง BIT5 - BIT0 เมื่อกำหนดแอดเดรสแล้วก็ทำการเขียนข้อมูลใน CG RAM โดยลักษณะบิทต่อบิท บนจอ 1 ตัวอักษร (1 ตัวอักษรใช้ข้อมูล 8 ไบท์) เมื่อเขียนข้อมูลลงใน CG RAM แล้ว เวลาที่ใช้งานก็ทำการเขียนข้อมูลลงใน DD RAM


DATA BUS
BAD0
BAD1
BAD2
BAD3
BAD4
BAD5
BAD6
BAD7
D0
D1
D2
D3
D4
D5
D6
D7
RD
WR
A0
A1
RESET
CS
I/O
BA0
BA1
8255:1
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
74LS245
DIR
74LS243
GAB
GBA
ADDRESS BUS
CONTROL BUS
VCC
10K
Adjust Intensity
L
C
D
MCS-51 TRAINING BOARD
Size
A
Document Number
Display
REV
Date:
March 17, 1992
Sheet 2 of

หลักการทำงานของหน่วยแสดงผล

ใช้พอร์ท A ตั้งแต่ PA0-PA7 เป็นตัวส่งข้อมูลให้ LCD เพื่อกำหนดการเลือกฟอร์นตัวอักษรออกมาแสดงผลทางจอภาพ โดยข้อมูลทั้ง 8 บิท จะถูกส่งผ่าน 74LS245 ซึ่งเป็นตัวทำหน้าที่รับส่งข้อมูลขนาด 8 บิท เมื่อจัดสัญญาน"0" เข้าที่ขา G และสัญญาน "1"ให้ขา DIR ข้อมูลจะถูกส่งผ่าน 74LS245 ไปยัง LCD

ใช้พอร์ท B ตั้งแต่ PB0-PB2 เป็นตัวส่งสัญญานไปควบคุมการทำงานของLCD สัญญานควบคุมมี 3 สัญญานคือ

-สัญญานเลือกรีจิสเตอร์ ส่งมาจาก PB0

-สัญญานกำหนดการเขียนหรืออ่าน ส่งมาจาก PB1

-สัญญานอินาเบิ้ล ส่งมาจาก PB2

ทั้ง 3 สัญญาน จะถูกส่งผ่าน 74LS243 ซึ่งเป็นตัวรับส่งข้อมูลขนาด 4 บิท ผ่านออกไปทางขา 4,5,6 ของ LCD

ที่เหลืออยู่ 3 ขาคือขา 1,2,3 ของ LCD จะต่อกับตัวต้านทานปรับค่าได้ขนาด 10 k เพื่อใช้ในการปรับความเข้มชัดของ LCD

การทดสอบ

เมื่อทำการทดสอบฮาร์ดแวร์ภายหลังจากลงอุปกรณ์ต่างๆเรียบร้อยแล้วนั้น ไม่สามารถที่จะทำงานได้ทันที ทั้งนี้ส่วนมากก็จะเกิดจากการที่ต่อวงจรในบางจุดไม่เรียบร้อยเท่าที่ควร เช่น บัดกรีไม่สนิทดีทำให้จุดที่ควรจะเชื่อมต่อเกิดการเปิดวงจรขึ้นมา ลายวงจรที่ได้จากการเขียนโดยใช้ซอฟแวร์สมาร์ทเวิร์คตกหล่นบางเส้นไป แต่ทั้งหมดนี้ก็สามารถแก้ไขได้เมื่อทำการตรวจเช็ค

ส่วนที่เป็นปัญหาที่ต้องใช้เวลาในการแก้ไขนานก็อยู่ที่ขั้นตอนการเขียนมอนิเตอร์โปรแกรมเพื่อใช้ควบคุมการทำงานของระบบทั้งหมดที่จัดสร้างขึ้นมา เหตุที่ต้องใช้เวลานานก็เพราะว่า การทำงานของไมโครโปรเซสเซอร์ที่ต่อร่วมอยู่กับอุปกรณ์ภายนอกนั้น ต้องทำให้จังหวะเวลาของซีพียูไม่ถูกรบกวน ในขณะเดียวกันต้องทำให้สอดคล้องกับจังหวะเวลาของอุปกรณ์อื่นอย่างพอดีด้วย

สรุปและวิจารณ์

จากการทำบอร์ดนี้ขึ้นมา ตั้งแต่เริ่มต้นจนกระทั่งถึงขั้นตอนการทดสอบ เมื่อประเมินผลงานทั้งหมดแล้วค่อนข้างจะน่าพอใจ อาจมีปัญหาบ้างในบางเรื่อง ต้องใช้เวลาในการแก้ไข มากบ้างน้อยบ้าง แล้วแต่ว่าปัญหาที่เกิดขึ้นนั้น จะได้เคยเจอมาบ้างหรือยัง ส่วนใหญ่แล้วปัญหาที่เกิดขึ้นเหล่านี้ สามารถพบเห็นได้ในตำราทั่วๆไปที่เกี่ยวกันระบบคอมพิวเตอร์ มีเพียงส่วนน้อยเท่านั้น ที่หาทางแก้ไข หรือคำอธิบายไม่ได้ว่า ทำไมถึงเป็นเช่นนั้น ต้องขอคำแนะนำจากอาจารย์ต่างๆ ไม่เฉพาะแต่อาจารย์ที่ปรึกษาเท่านั้น ในบางกรณี อาจมีปัญหาเกี่ยวกับอุปกรณ์ที่นำมาใช้ในการจัดทำโครงงาน อาทิเช่น การเชื่อมต่อ EPROM EMULATOR ระหว่างการเขียนมอนิเตอร์ โปรแกรมเกิดปัญหา คือ ทำการส่งข้อมูลออกมาไม่ได้ ภายหลังที่ปรึกษากับบริษัทที่ทำการผลิตอีพรอมอีมูเลเตอร์ตัวนี้ขึ้นมาแล้วจึงรู้ว่าแท้ที่จริงแล้วพอร์ทอนุกรมของคอมพิวเตอร์ที่ใช้งานอยู่ตั้งแต่แรกใช้งานไม่ได้ อีกกรณีหนึ่งก็คือ การแสดงผลออกทางจอแอลซีดี ซึ่งพยายามหาจุดผิดเท่าไรก็ไม่เจอ จึงไปปรึกษาที่บริษัทจึงรู้ว่าแอลซีดีใช้งานไม่ได้อีกเช่นกัน ปัญหาต่างๆเหล่านี้คิดว่าไม่น่าจะเกิดขึ้นแต่ก็เกิดมาแล้ว ทำให้รู้ว่าการทดสอบที่ได้ผลออกมาในทำนองที่แสดงออกไปในแนวโน้มว่าวงจรทำงานไม่ได้ แต่ที่จริงวงจรใช้งานได้ เป็นที่อุปกรณ์ที่นำมาช่วยทดสอบเกิดปัญหาขึ้นเอง

การใช้เครื่อง MCS-51 TRAINING BOARD

การใช้งานไมโครโปรเซสเซอร์ที่นำมาทำเป็นซิงเกิ้ลบอร์ดนั้น มักจะอยู่ในรูปของการเขียนโปรแกรมขึ้นมาแล้วทำการป้อนเข้าไปรันโดยใช้ซีพียูของเครื่อง ทำการรับค่าข้อมูลที่ต้องการเข้ามา จากนั้นก็จัดการกำหนดเอาท์พุทที่เหมาะสมกับข้อมูลนั้น นั่นคือบอร์ดจะต้องสามารถใช้ในการป้อนโปรแกรม การรันโปรแกรมที่ป้อนเข้าไป และการแสดงผลที่ได้ออกมา บอร์ดที่จัดทำขึ้นมาก็เช่นกัน มีคำสั่งต่างๆสร้างขึ้นมารองรับการป้อนโปรแกรมลักษณะต่างๆ การรันลักษณะต่างๆ ดังรายละเอียดต่อไปนี้

แผงคีย์บอร์ดประกอบด้วย ตัวอักษรภาษาอังกฤษ สัญลักษณ์พิเศษที่มีใช้ในภาษาแอสเซมบลีเฉพาะของซีพียูตระกูล MCS-51 เช่น #,@,$ และตัวเลขตั้งแต่ 0-9

เมื่อทำการเปิดเครื่องขึ้นมาโดยการต่อแหล่งจ่ายไฟเข้าสู่บอร์ด บอร์ดจะทำการรีเซ็ทตัวเอง แล้วจึงเข้าสู่ส่วนที่เป็นมอนิเตอร์โปรแกรมโดยอัตโนมัติ รอการกดคีย์ต่างๆเหล่านี้ เพื่อให้บอร์ดทำงานตามฟังก์ชันที่กำหนด คือ

คำสั่ง "M"

ใช้สำหรับการอ่านและเขียนข้อมูลในหน่วยความจำ ในส่วนที่เป็น DATA MAMORY และ PROGRAM MEMORY โดยเมื่อกดคีย์ เครื่องจะแสดงดังนี้

"M" ป้อน ADDRESS ที่ต้องการ จากนั้นกดคีย์ ENTER

เครื่องจะเข้าสู่การติดต่อกับหน่วยความจำทั้งสองชนิดทันที

คำสั่ง "R"

ใช้สำหรับการอ่านและเขียนข้อมูลในหน่วยความจำ ในส่วนที่เป็น INTERNAL RAM โดยเมื่อกดคีย์ เครื่องจะแสดงดังนี้

"R" ป้อน ADDRESS ที่ต้องการ จากนั้นกดคีย์ ENTER

เครื่องจะเข้าสู่การ MEMORY ทันที

คำสั่ง "G" ใช้สำหรับการ RUN PROGRAM โดยเมื่อกดคีย์ เครื่องจะแสดงดังนี้

"G" ป้อน ADDRESS ที่ต้องการ จากนั้นกดคีย์ ENTERเครื่องจะเข้าสู่การ RUN ทันที

คำสั่ง "S" ใช้สำหรับการ RUN PROGRAM ทีละ STEP โดยเมื่อกดคีย์ เครื่องจะแสดงดังนี้

"S" ป้อน ADDRESS ที่ต้องการ จากนั้นกดคีย์ ENTERเครื่องจะเข้าสู่การ STEP ทันที

คำสั่ง "F" ใช้สำหรับการอ่านและเขียนข้อมูลในหน่วยความจำ ในส่วนที่เป็น SPECIAL FUNCTION REGISTER ซึ่งอยู่ในภายในตัว 80C31 โดยเมื่อกดคีย์ เครื่องจะแสดงดังนี้

"F" ป้อน ADDRESS ที่ต้องการ จากนั้นกดคีย์ ENTER เครื่องจะเข้าสู่การ MEMORY ทันที

ชุดคำสั่งของ MCS-51

ชุดคำสั่งของ MCS-51 มีทั้งสิ้น 111 คำสั่ง ประกอบด้วย

คำสั่งขนาด 1 ไบท์ จำนวน 49 คำสั่ง

คำสั่งขนาด 2 ไบท์ จำนวน 45 คำสั่ง

คำสั่งขนาด 3 ไบท์ จำนวน 17 คำสั่ง

รูปแบบคำสั่งของออปโค้ดประกอบด้วยคำสั่งของนิวมอนิค ที่ตามด้วยโอเปอร์แรนด์ที่เป็นตัวรับการถ่ายทอด แหล่งกำเนิดในฟิลด์โอเปอร์แรนด์นี้จะเป็นแบบข้อมูลคงที่หรือตามแบบการใช้โหมดการกำหนดตำแหน่งเลขที่อยู่

ตามการออกแบบของอินเทล การกำหนดแอดเดรสหลายไบท์และตัวโอเปอร์แรนด์ข้อมูลจะเก็บไบท์ที่มีความสำคัญน้อยต่ำไว้ที่แอดเดรสตำแหน่งสูง และไบท์ที่มีความสำคัญน้อยสูงไว้ที่แอดเดสตำแหน่งต่ำ

ชุดคำสั่งของ MCS-51 ถูกแบ่งตามลักษณะการทำงานได้เป็น 4 กลุ่ม คือ

กลุ่มการถ่ายเทข้อมูล

กลุ่มคณิตศาสตร์

กลุ่มตรรกศาสตร์

กลุ่มการควบคุมการถ่ายเท

รายละเอียดชุดคำสั่ง MCS-51

ACALL addr 11

องค์ประกอบ: Absolute Call

จำนวนไบท์: 2

วัฏจักรแมชชีน: 2

การทำงาน:

(PC) <----- (PC)+2

(SP) <----- (SP)+1

((SP)) <----- (PC_{7-0})

(SP) <----- (SP)+1

((SP)) <----- (PC_{15-8})

(PC_{10-0}) <----- page address

ADD A,<src-byte>

องค์ประกอบ: add

ADD A,Rn ;รีจิสเตอร์

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A)+(Rn)

ADD A,direct ;โดยตรง

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A)+(direct)

ADD A,@Ri ;โดยอ้อมรีจิสเตอร์

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A)+((Ri))

ADD A,#data ;โดยทันที

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A)+#data

ADDC A,<src-byte>

องค์ประกอบ: Add with Carry

ADDC A,Rn

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน: ADDC

ADDC A,direct

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A)+(C)+(direct)

ADDC A,@Ri

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A)+(C)+((Ri))

ADDC A,#data

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A)+(C)+#data

AJUMP addr11

องค์ประกอบ: Absolute Jump

จำนวนไบท์: 2

วัฏจักรแมชชีน: 2

การทำงาน:

(PC) <----- (PC)+2

(PC_{10-0}) <----- page address

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

องค์ประกอบ: ANL

ANL A,Rn

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A) ^ (Rn)

ANL A,direct

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A) ^ (direct)

ANL A;@Ri

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A) ^ (Ri)

ANL A,#data

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

. (A) <----- (A) ^ #data

ANL direct,A

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(direct) <----- (direct) ^ (A)

ANL direct,#data

จำนวนไบท์: 3

วัฏจักรแมชชีน: 2

การทำงาน:

(direct) <----- (direct) ^ #data

ANL C,<src-byte

องค์ประกอบ: ตรรก AND สำหรับตัวแปรขนาดบิท

ANL C,bit

จำนวนไบท์: 2

วัฏจักรแมชชีน: 2

การทำงาน:

(C) <----- (C) ^ (bit)

ANL C,/bit

จำนวนไบท์: 2

วัฏจักรแมชชีน: 2

การทำงาน:

(C) <----- (C) ^ /(bit)

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

องค์ประกอบ: เปรียบเทียบและกระโดดหากไม่เท่ากัน

CJNE A,direct,rel ;ACC.กับโหมดโดยตรง

จำนวนไบท์: 3

วัฏจักรแมชชีน: 2

การทำงาน:

(PC) <----- (PC)+3

IF (direct) < (A)

THEN (PC) <----- (PC)+rel and (C) <----- 0

IF (direct) > (A)

THEN (PC) <----- (PC)+rel and (C) <----- C

CJNE A,#data,rel ;ACC.กับโหมดโดยทันที

จำนวนไบท์: 3

วัฏจักรแมชชีน: 2

การทำงาน:

(PC) <----- (PC)+3

IF #data < A

THEN (PC) <----- +rel and (C) <----- 0

IF #data > A

THEN (PC) <----- +rel and (C) <----- 1

CLNE Rn,#data,rel ;โหมดรีจิสเตอร์กับโหมดโดยทันที

จำนวนไบท์: 3

วัฏจักรแมชชีน: 2

การทำงาน:

(PC) <----- (PC)+3

IF #data < (Rn)

THEN (PC) <----- (PC)+rel and (C) <----- 0

IF #data > (Rn)

THEN (PC) <----- (PC)+rel and (C) <----- 1

CJNE @Ri,#data,rel ;โหมดโดยอ้อมกับโหมดโดยทันที

จำนวนไบท์: 3

วัฏจักรแมชชีน: 2

การทำงาน:

(PC) <----- (PC)+3

IF #data < (Ri)

THEN (PC) <----- (PC)+rel and (C) <----- 1

IF #data > (Rn)

THEN (PC) <----- (PC)+rel and (C) <----- 0

CLR A

องค์ประกอบ: เคลียร์แอคคิวมูเลเตอร์

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- 0

CLR bit

องค์ประกอบ: เคลียร์บิท

CLR C

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(C) <----- 0

CLR bit

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(bit) <----- 0

CPL A

องค์ประกอบ: Complement Accumulator

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A)

CPL bit

องค์ประกอบ: Complement Bit

CPL C

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(C) <----- (C)

CPL bit

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(BIT) <----- (BIT)

DA A

องค์ประกอบ: การปรับค่าในแอคคิวมูเลเตอร์เป็นเลขฐานสิบหลังทำคำสั่งบวก

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

DA-contents of Accumulator are BCD

IF [(A_{3-0}) >9] v [(AC)=1]

THEN (A_{3-0}) <----- $(A_{3-0})+6$

IF [(A_{7-4}) >9] v [(AC)=1]

THEN (A_{7-4}) <----- $(A_{7-4})+6$

DEC byte

องค์ประกอบ: Decrement

DEC A

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A)-1

DEC Rn

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(Rn) <----- (Rn)-1

DEC direct

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(direct) <----- (direct)-1

DEC @Ri

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

((Ri)) <----- -1

DIV AB

องค์ประกอบ: Divide

จำนวนไบท์: 1

วัฏจักรแมชชีน: 4

การทำงาน:

$(A)_{15-8}$

$(B)_{7-0}$

DJNZ <byte>,<rel-addr>

องค์ประกอบ: การลดค่าหนึ่งค่าและกระโดดถ้าตัวแปรที่ถูกกำหนดไม่เป็นศูนย์

DJNZ Rn,rel

จำนวนไบท์: 2

วัฏจักรแมชชีน: 2

การทำงาน:

(PC) <----- (PC)+2

(Rn) <----- (Rn)-1

IF (Rn) > 0 or (Rn) < 0

THEN

(PC) <----- (PC)+rel

DJNZ direct,rel

จำนวนไบท์: 3

วัฏจักรแมชชีน: 2

การทำงาน:

(PC) <----- (PC)+2

(direct) <----- (direct)-1

IF (direct) > 0 or (direct) < 0

THEN

(PC) <----- (PC)+rel

INC <byte>

องค์ประกอบ: Increment

INC A

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A)+1

INC Rn

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(Rn) <----- (Rn)+1

INC direct

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(direct) <----- (direct)+1

INC @Ri

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

((Ri)) <----- +1

INC DPTR

องค์ประกอบ: Increment Data Pointer

จำนวนไบท์: 1

วัฏจักรแมชชีน: 2

การทำงาน:

(DPTR) <----- +1

JB bit,rel

องค์ประกอบ: กระโดดถ้าบิทที่ถูกกำหนดเป็นหนึ่ง

จำนวนไบท์: 3

วัฏจักรแมชชีน: 2

การทำงาน:

(PC) <----- (PC)+3

IF (bit)=1

THEN (PC) <----- (PC)+rel

JBC bit,rel

องค์ประกอบ: กระโดดถ้าบิทที่ถูกกำหนดเป็น 1 และจะเคลียร์บิทนั้น

จำนวนไบท์: 3

วัฏจักรแมชชีน: 2

การทำงาน:

(PC) <----- (PC)+3

IF (bit) = 1

THEN (bit) <----- 0

(PC) <----- (PC)+rel

JC rel

องค์ประกอบ: กระโดดถ้าบิททดเป็นหนึ่ง

จำนวนไบท์: 2

วัฏจักรแมชชีน: 2

การทำงาน:

(PC) <----- (A)+(DPTR)

JMP A,@A+DPTR

องค์ประกอบ: กระโดดโดยอ้อม

จำนวนไบท์: 1

วัฏจักรแมชชีน: 2

การทำงาน:

(PC) <----- (A)+(DPTR)

JNB bit,rel

องค์ประกอบ: กระโดดถ้าบิทที่ถูกกำหนดไม่เป็น 1

จำนวนไบท์: 3

วัฏจักรแมชชีน: 2

การทำงาน:

(PC) <----- (PC)+3

IF (bit) = 0

THEN (PC) <----- (PC)+rel

JNC rel

องค์ประกอบ: กระโดดถ้าบิททดไม่ถูกเซ็ท

จำนวนไบท์: 2

วัฏจักรแมชชีน: 2

การทำงาน:

(PC) <----- (PC)+2

IF (C) =0

THEN (PC) <----- (PC)+rel

JNZ rel

องค์ประกอบ: กระโดดถ้าค่าในแอคคิวมูเลเตอร์ไม่เป็น 0

จำนวนไบท์: 2

วัฏจักรแมชชีน: 2

การทำงาน:

(PC) <----- (PC)+2

IF (A) # 0

THEN (PC) <----- (PC)+rel

JZ rel

องค์ประกอบ: กระโดดถ้าค่าในแอคคิวมูเลเตอร์เป็น 0

จำนวนไบท์: 2

วัฏจักรแมชชีน: 2

การทำงาน:

(PC) <----- (PC)+2

IF (A) = 0

THEN (PC) <----- (PC)+rel

LCALL addr 16

องค์ประกอบ: Long Call

จำนวนไบท์: 3

วัฏจักรแมชชีน: 2

การทำงาน:

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- ((Ri))

MOV A,#data

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- #data

MOV Rn,A

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(Rn) <----- (A)

MOV Rn,direct

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(Rn) <----- (direct)

MOV Rn,#data

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(Rn) <----- #data

MOV direct,A

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(direct) <----- (A)

MOV direct,Rn

จำนวนไบท์: 2

วัฏจักรแมชชีน: 2

การทำงาน:

(direct) <----- (Rn)

MOV direct,direct

จำนวนไบท์: 3

วัฏจักรแมชชีน: 2

การทำงาน:

(direct) <----- (direct)

MOV direct,@Ri

จำนวนไบท์: 2

วัฏจักรแมชชีน: 2

การทำงาน:

(direct) <----- ((Ri))

MOV direct,#data

จำนวนไบท์: 3

วัฏจักรแมชชีน: 2

การทำงาน:

(direct) <----- #data

MOV #Ri,A

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

((Ri)) <----- (A)

MOV #Ri,direct

จำนวนไบท์: 2

วัฏจักรแมชชีน: 2

การทำงาน:

((Ri)) <----- (direct)

MOV #Ri,#data

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

((Ri)) <----- #data

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

องค์ประกอบ: Move Bit Data

MOV C,bit

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(C) <----- (bit)

MOV bit,C

จำนวนไบท์: 2

วัฏจักรแมชชีน: 2

การทำงาน:

(bit) <----- (c)

MOV DPTR,#data16

องค์ประกอบ: โหลดค่าคงที่ขนาด 16 บิทเข้าที่ตัวชี้ข้อมูล

จำนวนไบท์: 3

วัฏจักรแมชชีน: 2

การทำงาน:

(DPTR) <----- #data

DPH[]DPL <----- #data$_{15-8}$[]#data$_{7-0}$

MOVC A,@A+<base-reg>

องค์ประกอบ: ย้ายค่ารหัสไบท์ของหน่วยความจำโปรแกรม

MOVC A,@A+DPTR

จำนวนไบท์: 1

วัฏจักรแมชชีน: 2

การทำงาน:

(A) <----- (A)+(DPTR)

MOVC A,@A+PC

จำนวนไบท์: 1

วัฏจักรแมชชีน: 2

การทำงาน:

(PC) <----- (PC)+1

(A) <----- ((A)+(PC))

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

องค์ประกอบ: mov ข้อมูลจากหน่วยความจำภายนอก

MOVX A,@Ri

จำนวนไบท์: 1

วัฏจักรแมชชีน: 2

การทำงาน:

(A) <----- ((Ri))

MOVX A,@DPTR

จำนวนไบท์: 1

วัฏจักรแมชชีน: 2

การทำงาน:

(A) <----- ((DPTR))

MOVX @Ri,A

จำนวนไบท์: 1

วัฏจักรแมชชีน: 2

การทำงาน:

((Ri)) <----- (A)

MOVX @DPTR,A

จำนวนไบท์: 1

วัฏจักรแมชชีน: 2

การทำงาน:

(DPTR) <----- (A)

MUL AB

องค์ประกอบ: Multiply

จำนวนไบท์: 1

วัฏจักรแมชชีน: 4

การทำงาน:

$(A)_{7-0}$ <----- (A)x(B)

$(B)_{15-0}$

NOP

องค์ประกอบ: No การทำงาน

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(PC) <----- (PC)+1

ORL <dest-byte>,<src-byte>

องค์ประกอบ: Logical-Or for byte Variable

ORL A,Rn

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A) v (Rn)

ORL A,direct

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A) v (direct)

ORL A,@Ri

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A) v ((Ri))

ORL A,#data

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A) v #data

ORL direct,A

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(direct) <----- (direct) v (A)

ORL direct,#data

จำนวนไบท์: 3

วัฏจักรแมชชีน: 2

การทำงาน:

(direct) <----- (direct) v #data

ORL C,<src-bit>

องค์ประกอบ: กระทำตรรก OR ด้วยขนาดตัวแปรบิท

ORL C,bit

จำนวนไบท์: 2

วัฏจักรแมชชีน: 2

การทำงาน:

(C) <----- (C) v (BIT)

POP direct

องค์ประกอบ: Pop from Stack

จำนวนไบท์: 2

วัฏจักรแมชชีน: 2

การทำงาน:

(direct) <----- ((SP))

(SP) <----- (SP)-1

PUSH direct

องค์ประกอบ: Push onto Stack

จำนวนไบท์: 2

วัฏจักรแมชชีน: 2

การทำงาน:

(SP) <----- (SP)+1

((SP)) <----- (direct)

RET

องค์ประกอบ: กลับมาจากโปรแกรมย่อย

จำนวนไบท์: 1

วัฏจักรแมชชีน: 2

การทำงาน:

(PC_{15-8}) <----- ((SP))

(SP) <----- (SP)-1

(PC_{7-0}) <----- ((SP))

(SP) <----- (SP)-1

RETI

องค์ประกอบ: กลับมาจากการอินเทอร์รัพ

จำนวนไบท์: 1

วัฏจักรแมชชีน: 2

การทำงาน:

(PC_{15-8}) <----- ((SP))

(SP) <----- (SP)-1

(PC_{7-0}) <----- ((SP))

(SP) <----- (SP)-1

RL A

องค์ประกอบ: เลื่อนแอคคิวมูเลเตอร์ไปทางซ้าย

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

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

$(A_n) \leftarrow (A_7)$

RLC A

องค์ประกอบ: เลื่อนแอคคิวมูเลเตอร์ไปทางซ้าย และผ่านแฟลกตัวทด

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

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

$(A_0) \leftarrow (C)$

$(C) \leftarrow (A_7)$

RR A

องค์ประกอบ: เลื่อนแอคคิวมูเลเตอร์ไปทางขวา

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

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

$(A_7) \leftarrow (A_0)$

RRC A

องค์ประกอบ: เลื่อนแอคคิวมูเลเตอร์ไปทางขวา และผ่านแฟลกตัวทด

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(A_n) <----- (A_{n+1}) ; n=0-6

(A_7) <----- (C)

(C) <----- (A_0)

SETB <bit>

องค์ประกอบ: เซตบิท

SETB C

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(C) <----- 1

SETBB bit

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(bit) <----- 1

SJUMP rel

องค์ประกอบ: Shot Jump

จำนวนไบท์: 2

วัฏจักรแมชชีน: 2

การทำงาน:

(PC) <----- (PC)+2

(PC) <----- (PC)+rel

SUBB A,<src-byte>

องค์ประกอบ: ลบตัวแปรที่ถูกกำหนดและบิททด ออกจากค่าในแอคคิวมูเลเตอร์

SUBB A,Rn

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A)-(C)-(Rn)

SUBB A,direct

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A)-(C)-(direct)

SUBB A,@Ri

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A)-(C)-((Ri))

SUBB A,#data

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A)-(C)-#data

SWAP A

องค์ประกอบ: swapนิบเบิ้ล ในแอคคิวมูเลเตอร์

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(A_{3-0}) <-----> (A_{7-4}),

XCH A,<byte>

องค์ประกอบ: แลกเปลี่ยนข้อมูลแอคคิวมูเลเตอร์กับข้อมูลตัวแปร

XCH A,Rn

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <-----> (Rn)

XCH A,direct

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <-----> (direct)

XCH A,@Ri

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <-----> ((Ri))

XCHD A,@Ri

องค์ประกอบ: Exchange Digit

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(A_{3-0}) <-----> $((Ri_{3-0}))$

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

องค์ประกอบ: Logical Exclusive-OR for byte Variable

XRL A,Rn

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A) v (Rn)

XRL A,direct

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A) v (direct)

XRL A,@Ri

จำนวนไบท์: 1

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A) v ((Ri))

XRL A,#data

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(A) <----- (A) v #data

XRL direct,A

จำนวนไบท์: 2

วัฏจักรแมชชีน: 1

การทำงาน:

(direct) <-----> (direct) v (A)

XRL direct,#data

จำนวนไบท์: 3

วัฏจักรแมชชีน: 2

การทำงาน:

(direct) <-----> (direct) v #data

```
                        ORG     0000H
;*****************************************************************************
;*********                 MAIN PROGRAM                              **********
                    ************
;*****************************************************************************
                        DS      8
                        DS      24
                        DS      2
BITADD:                 DS      3
DISBUF:                 DS      8
HEXBUF:                 DS      3
SCANKM:                 DS      1
SCANAM:                 DS      1
SCANAR:                 DS      1
SCANRC:                 DS      1
SCANFC:                 DS      1
RAMTMM:
DEMUFS:
INPBKH:                 DS      1
DEMRND:
INPBKL:                 DS      1
DEMCNT:
INPBKC:
CHANAD:
FINDAD:
SEGDEM:
SAVENO:
DEMSOU:
DEBUGM:                 DS      1
BRKMEM:
DEMSCO:
TOOLHX:                 DS      1
TOOLMM:
TEMADD:
MEMADD:                 DS      2
IXDATA:                 DS      2
PMMADD:
DMMADD:                 DS      2
INRADD:                 DS      1
RUNADD:                 DS      2
BRKADD:                 DS      2
STTADD:                 DS      2
ENDADD:                 DS      2
DESADD:                 DS      2
HEXDAT:                 DS      1
S2BAUD:                 DS      1
POWMEM:                 DS      1
VTSP:                   DS      1
VTDPL:                  DS      1
VTDPH:                  DS      1
VTTCON:                 DS      1
VTTMOD:                 DS      1
VTTL0:                  DS      1
VTTL1:                  DS      1
```

```
SCANOL                  EQU   19H
AUTOIF                  EQU   1AH
FUNC2F                  EQU   1BH
DEBUGF                  EQU   1CH
TRACEF                  EQU   1DH
MULFG                   EQU   1EH
INPFIR                  EQU   1FH
BRKFAG                  EQU   20H
SEFLAG                  EQU   21H
INPBFG                  EQU   22H
DECFAG                  EQU   22H
GETOKF                  EQU   22H
EEPFAG                  EQU   23H
SENDEN                  EQU   23H
DUMPBM                  EQU   23H
POWFAG                  EQU   23H
STMFAG                  EQU   23H
DEMWIN                  EQU   23H
RELTMM                  EQU   23H
;*************************************************************
                        ORG     0000H
                        LJMP    MON
IOVEC:                  PUSH    IE0ADD+1
                        PUSH    IE0ADD
                        RET
                        DB      0FFH,0FFH,0FFH
TF0VEC:                 PUSH    TF0ADD+1
                        PUSH    TF0ADD
                        RET
                        DB      0FFH,0FFH,0FFH
IE1VEC:                 JB      DEBUGF,IE1VEC1
                        LJMP    STEPR
IE1VEC1:                RET
                        DB      0FFH
TF1VEC:                 DB      0FFH,0FFH,0FFH
                        DB      0FFH,0FFH,0FFH
                        DB      0FFH,0FFH
RTIVEC:                 PUSH    RTIADD+1
                        PUSH    RTIADD
                        RET
                        DB      0FFH,0FFH,0FFH
TF2VEC:                 DB      0FFH,0FFH,0FFH
                        DB      0FFH,0FFH,0FFH
                        DB      0FFH,0FFH
BRKVEC:                 LJMP    BRK
ENDVEC:                 LJMP    ENDV

MON:                    LCALL   LDELAY
                        MOV     R0,#08H             ;CLEAR  INT-RAM
                        MOV     R2,#77H
RES:                    MOV     @R0,#0
                        INC     R0
```

```
DEBUGS:     MOV     A,DEBUGM        ;LOAD HEXBUF+2
            MOV     DPH,VTDPH       ;@DPTR
            MOV     DPL,VTDPL
            MOVX    A,@DPTR
            MOV     HEXBUF+2,A
            RET

;**********************STEP MAIN**********************************
STEP:       MOV     DPTR,#0F801H    ;INT1 LOW
            MOVX    A,@DPTR
            CLR     ACC.3
            MOVX    @DPTR,A
            SETB    DEBUGF
            SETB    EX1
            SETB    EA
            NOP                     ;INTERRUPT ACTIVE
            CLR     DEBUGF          ;ON INTERRUPT ROTINE
            LCALL   EXIT            ;EXIT PROCESS
            MOV     SP,VTSP
            PUSH    RUNADD+1        ;PC
            PUSH    RUNADD
            RETI                    ;GO

;**********************STEPR MAIN*********************************
STEPR:      MOV     VTSP,SP         ;RETURN BY INT1
            MOV     SP,#STACK       ;LOAD PC
            LCALL   RETS
            MOV     R0,VTSP         ;DEC STACK
            MOV     RUNADD,@R0      ;LOAD PC(H)
            DEC     R0
            MOV     RUNADD+1,@R0    ;LOAD PC(L)
            DEC     R0
            MOV     VTSP,R0
            MOV     DPTR,#0F801H    ;INT1 HIGH
            MOVX    A,@DPTR
            SETB    ACC.3
            MOVX    @DPTR,A
            CLR     EA
            CLR     EX1
            CLR     SCANF3
            CLR     SCANF5
            CLR     SCANFF
            SETB    DEBUGF
            MOV     DPH,RUNADD      ;CHECK ENDVEC.
            MOV     DPL,RUNADD+1
            MOV     R2,#00H
            MOV     R3,#36H
            LCALL   DPCOM
            JNZ     STEPR2
            MOV     DPTR,#ENDV1     ;END PROGRAM
            PUSH    DPL
            PUSH    DPH
            RETI
```

```
STEPR2:     MOV     R2,#80H         ;CHECK MONITOR ADDRE
            MOV     R3,#00H
            LCALL   DPCOM
            JNZ     STEPR3
            LCALL   STKA            ;CALL MONTOR
            LCALL   EXIT            ;EXIT PROCESS
            MOV     SP,VTSP
            PUSH    RUNADD+1
            PUSH    RUNADD
            RET                     ;RETURN TO STEPR
STEPR3:     MOV     DPTR,#DEBUGX    ;STEP
            PUSH    DPL
            PUSH    DPH
            RETI

;**********************STKA SUB.**********************************
;STACK ANALYSIS

STKA:       MOV     R1,VTSP         ;RET ADDRESS PROGRAM
STKA1:      MOV     A,R1
            MOV     R0,A
            MOV     DPH,@R0
            DEC     R0
            MOV     DPL,@R0
            LCALL   DPDEC
            LCALL   DPDEC
            CLR     A               ;COMPARE 00H
            MOVC    A,@A+DPTR
            CJNE    A,#00H,STKA2
            LCALL   DPDEC
            CLR     A
            MOVC    A,@A+DPTR
            CJNE    A,#12H,STKA2
            SJMP    STKA3           ;FOUND
STKA2:      DEC     R1              ;NEXT SEARCH
            CJNE    R1,#07H,STKA1
            MOV     SP,#STACK
            LJMP    MON             ;NOT FOUND
STKA3:      MOV     B,R1            ;USER-STACK INSERT
            MOV     A,VTSP          ;2 BYTE
            MOV     R0,A
            INC     A
            INC     A
            MOV     R1,A
            MOV     VTSP,A

STKA31:     MOV     A,R0
            CJNE    A,B,$+6         ;COMPARE A-B
            LJMP    STKA4
            MOV     A,@R0           ;MOVE @R0->@R1
            MOV     @R1,A
            DEC     R0
```

```
                        DEC     R1
                        SJMP    STKA31
STKA4:                  INC     R0              ;INSERT STEPR FOR RET
                        MOV     @R0,#LOW STEPR
                        INC     R0
                        MOV     @R0,#HIGH STEPR
                        RET
;*********************GO & RUN MAIN*********************************

GO:                     LCALL   CLEARSCREEN
                        LCALL   LDELAY
                        MOV     DPTR,#TABLE5
                        LCALL   WLINE3
                        LCALL   LDELAY
                        LCALL   INW
                        MOV     DPH,HEXBUF+0
                        MOV     DPL,HEXBUF+1
                        MOV     RUNADD,DPH
                        MOV     RUNADD+1,DPL
                        JB      INPBFG,GO
                        LCALL   RETS
                        MOV     DPH,RUNADD      ;CHECK PROGRAM NOT=00
                        MOV     DPL,RUNADD+1
                        CLR     A
                        MOVC    A,@A+DPTR
                        CJNE    A,#0,RUN1
                        LJMP    DEBUGX
RUN1:                   LCALL   EXIT            ;EXIT PROCESS
                        MOV     SP,VTSP
                        PUSH    RUNADD+1        ;PC
                        PUSH    RUNADD
                        RETI                    ;GO

;*********************ENDV MAIN*************************************
;RETURN TO MON
ENDV:                   MOV     VTSP,SP         ;RET FROM USER PROGRAM
                        MOV     SP,#STACK
                        LCALL   RETS
ENDV1:                  LCALL   BRKCHK
                        MOV     RUNADD,#80H
                        MOV     RUNADD+1,#0
                        CLR     SCANBF
                        CLR     EA
                        CLR     EX1
                        MOV     A,#LOW MON
                        PUSH    ACC
                        MOV     A,#HIGH MON
                        PUSH    ACC
                        RET                     ;TO MON

;*********************BRK MAIN**************************************
;SOFTWARE-BREAK RETURN
BRK:                    MOV     VTSP,SP         ;RETURN BY SOFTWARE-BRAE
```

```
                        MOV     SP,#STACK
                        LCALL   RETS
                        MOV     RUNADD,BRKADD
                        MOV     RUNADD+1,BRKADD+1
                        MOV     DPH,BRKADD      ;RETURN USER-DATA
                        MOV     DPL,BRKADD+1
                        MOV     A,BRKMEM        ;BYTE 0
                        MOVX    @DPTR,A
                        INC     DPTR
                        MOV     A,BRKMEM+1      ;BYTE 1
                        MOVX    @DPTR,A
                        INC     DPTR
                        MOV     A,BRKMEM+2      ;BYTE 2
                        MOVX    @DPTR,A
                        LCALL   DEBUGX
;*****************************************************************
;BREAK CHECK SUB. FOR RESET,ENDVEC,STEPR(HW-BREAK)
;MEAING-SET SW-BREAK BUT NOT PASS-BREAK-ADDRESS
BRKCHK:                 JB      BRKFAG,BRKCHK1
                        RET
BRKCHK1:                CLR     BRKFAG
                        MOV     DPH,BRKADD      ;RETURN USER-DATA
                        MOV     DPL,BRKADD+1
                        MOV     A,BRKMEM        ;BYTE 0
                        MOVX    @DPTR,A
                        INC     DPTR
                        MOV     A,BRKMEM+1      ;BYTE 1
                        MOVX    @DPTR,A
                        INC     DPTR
                        MOV     A,BRKMEM+2      ;BYTE 2
                        MOVX    @DPTR,A
                        RET

;**********************EXIT SUB.**********************************
;LOAD SFR & Rn FOR USER PROGRAM(EXCEPT SP)

EXIT:                   MOV     DPL,VTDPL       ;SFR
                        MOV     DPH,VTDPH
                        MOV     TCON,VTTCON
                        MOV     TMOD,VTTMOD
                        MOV     TL0,VTTL0
                        MOV     TL1,VTTL1
                        MOV     TH0,VTTH0
                        MOV     TH1,VTTH1
                        MOV     P1,VTP1
                        MOV     SCON,VTSCON
                        MOV     IE,VTIE
                        MOV     IP,VTIP
                        MOV     PSW,VTPSW
                        MOV     A,VTACC
                        MOV     B,VTB
                        MOV     R0,VTREG        ;REG
                        MOV     R1,VTREG+1
```

```
                MOV     DISBUF+5,#20
                MOV     DISBUF+6,#20
                MOV     DISBUF+7,#20
                MOV     DISBUF+8,#20
                LCALL   WLINE4
INTR2:          LCALL   SCAN
                CJNE    A,#23H,INTR4            ;DEC KEY
                DEC     INRADD
                LJMP    INTR1
INTR4:          CJNE    A,#24H,INTR44           ;INC KEY
                INC     INRADD
                LJMP    INTR1
INTR44:         CJNE    A,#12H,INTR46           ;ENT KEY
                CPL     AUTOIF
                LJMP    INTR1
INTR46:         CJNE    A,#13H,INTR5            ;BACK KEY
                CPL     SCANRF
                LJMP    INTR1
INTR5:          LCALL   GET2                    ;EDIT
                JB      GETOKF,INTR6
                LJMP    INTR2
INTR6:          MOV   R2,DISBUF+3               ;WRITE
                MOV   R3,DISBUF+4
                LCALL ATOH                      ;EXIT
                MOV   HEXBUF+1,A
                LCALL INTRWR
                INC     INRADD
                LJMP    INTR1

;*****INTRRD SUB*****
;READ DATA SUB
;IN=MEMADD
;OUT=A
;REG=A,R0,DPTR
INTRRD:         MOV     A,INRADD
                CJNE    A,#08H,$+3
                JNC     INTRRD2                 ;=>
                ADD     A,#VTREG                ;VIRTUAL INT-RAM
                MOV     R0,A
                MOV     A,@R0
                RET
INTRRD2:        MOV     R0,A                    ;REAL INT-RAM
                MOV     A,@R0
                RET

;*****INTRWR SUB*****
;WRITE DATA SUB
;IN=A,MEMADD
;REG=A,R0,DPTR

INTRWR:         PUSH    ACC                     ;-[
                MOV     A,INRADD
                CJNE    A,#08H,$+3
```

```
                CJNE   A,#08H,$+3
                JNC    INTRWR2                ;=>
                ADD    A,#VTREG               ;VIRTUAL INT-RAM
                MOV    R0,A
                POP    ACC                    ;-]
                MOV    @R0,A
                RET
INTRWR2:        MOV    R0,A                   ;REAL INT-RAM
                POP    ACC                    ;-]
                MOV    @R0,A
                RET
;*****INH SUB*****
;INPUT 2 DIGIT HEX
;IN = A HEX
;   = R2 CHAR
;OUT= A HEX
;   =INPBFG
;REG=ALL
INH:            SETB INPFIR
INH1:           LCALL SCAN                    ;WAIT KEY
                MOV  R2,A
                CLR  C
                SUBB A,#10H
                MOV  A,R2
                JNC  INH5                     ;>F
                LCALL HTOAS                   ;CHANGE TO SEGMENT CODE
                MOV  R0,A
                JB   INPFIR,INH3              ;FIRST DIGIT PROCESS
                MOV  R3,A
                MOV  DISBUF+1,R0
INH2:           LCALL   WRITECHAR
                SETB INPFIR
                SJMP INH1
INH3:           MOV  DISBUF+0,R0
                MOV  DISBUF+1,#20
                MOV  DISBUF+2,#20
                MOV  DISBUF+3,#20H
                MOV  DISBUF+4,#20H
                MOV  DISBUF+5,#20H
                LCALL WLINE31
                MOV  A,#4
                LCALL GOTOXY
                CLR  INPFIR
                LJMP INH1
INH5:           MOV  R2,DISBUF+0
                MOV  R3,DISBUF+1
                LCALL ATOH                    ;EXIT
                MOV  HEXBUF+0,A
                CLR  INPBFG
                RET
;******GET2 SUB*******
;GET HEX ON DIGIT 4,5
;IN = A KEY CODE 0-FH
```

```
;OUT= DISBUF
;    =GETOKF(2 DIGIT COMPLETE)
;REG= A,R2,DPTR
GET2:       CLR  GETOKF
            PUSH ACC
            CLR  C
            SUBB A,#10H              ;COMPARE
            POP  ACC
            JC   GET21
            RET

GET21:      MOV  R2,A
            LCALL HTOAS
            MOV  R2,A
            MOV  A,DISBUF+4
            ANL  A,#0DFH             ;AND DOT
            CJNE A,#0H,GET25
            MOV  A,R2
            MOV  DISBUF+4,A          ;LAST DIGIT
            MOV  A,#7
            LCALL GOTOXY
            MOV  A,DISBUF+4
            MOV  R0,A
            LCALL   WRITECHAR
            SETB GETOKF
            LCALL  DELAY
            RET
GET25:      MOV  DISBUF+3,R2         ;FIRST DIGIT
            MOV  A,R2
            MOV  DISBUF+4,#20H
            LCALL WLINE5
            RET


;*****DMEM MAIN*****
;DATA MEMORY EDIT
DMEM:       LCALL CLEARSCREEN
            LCALL LDELAY
            MOV   DPTR,#TABLE3
            LCALL WLINE3
            LCALL INW
            MOV   DPH,HEXBUF+0
            MOV   DPL,HEXBUF+1
            MOV   DMMADD,DPH
            MOV   DMMADD+1,DPL
            JB    INPBFG,DMEM
            MOV   MEMADD,DMMADD
            MOV   MEMADD+1,DMMADD+1

DMEM1:      MOV   DPH,MEMADD         ;EDIT LOOP
            MOV   DPL,MEMADD+1
          . MOV   HEXBUF,DPH
```

```
                MOV     HEXBUF+1,DPL
                MOVX    A,@DPTR
                MOV     HEXBUF+2,A              ;DATA MEM
                MOV     A,HEXBUF+0
                LCALL   HTOA
                MOV     DISBUF+0,R2
                MOV     DISBUF+1,R3
                MOV     A,HEXBUF+1
                LCALL   HTOA
                MOV     DISBUF+2,R2
                MOV     DISBUF+3,R3
                MOV     DISBUF+4,#20H
                MOV     A,HEXBUF+2
                LCALL   HTOA
                MOV     DISBUF+5,R2
                MOV     DISBUF+6,R3
                MOV     DISBUF+7,#20H
                LCALL   WLINE4
DMEM2:          LCALL   SCAN
                CJNE    A,#23H,DMEM4            ;DEC KEY
                MOV     DPH,MEMADD
                MOV     DPL,MEMADD+1
                LCALL   DPDEC
                MOV     MEMADD,DPH
                MOV     MEMADD+1,DPL
                MOVX    A,@DPTR
                LJMP    DMEM1
DMEM4:          CJNE    A,#24H,DMEM5            ;INC KEY
DMEM41:         MOV     DPH,MEMADD
                MOV     DPL,MEMADD+1
                INC     DPTR
                MOV     MEMADD,DPH
                MOV     MEMADD+1,DPL
                LJMP    DMEM1
    DMEM44:             CJNE    A,#12H,DMEM46   ;ENT KEY
                        CPL     AUTOIF
                        LJMP    DMEM1
    DMEM46:             CJNE    A,#13H,DMEM5    ;BACK KEY
                        CPL     SCANRF
                        LJMP    DMEM1
DMEM5:          LCALL   GET1
                JB      GETOKF,DMEM6
                LJMP    DMEM2
DMEM6:          MOV     R2,DISBUF+5
                MOV     R3,DISBUF+6
                LCALL   ATOH                    ;WRITE
                MOV     HEXBUF+2,A
                MOV     DPH,MEMADD
                MOV     DPL,MEMADD+1
                MOV     A,HEXBUF+2
                MOVX    @DPTR,A
                MOV     A,#0FFH
                MOVX    A,@DPTR
```

```
;*****INW SUB*****
;INPUT 2 DIGIT HEX
;IN = A HEX
;    = R2 CHAR
;OUT= A HEX
;   =INPBFG
;REG=ALL
INW:        SETB INPFIR
            MOV  R5,#3
            MOV  R4,#DISBUF+0

INW1:       LCALL SCAN                  ;WAIT KEY
            MOV  R2,A
            CLR  C
            SUBB A,#10H
            MOV  A,R2
            JNC  INW5                   ;>F
            LCALL HTOAS                 ;CHANGE TO SEGMENT CODE
            MOV  R0,A
            JB   INPFIR,INW3            ;FIRST DIGIT PROCESS
INW2:       LCALL   WRITECHAR
            INC  R4
            MOV  A,R4
            MOV  R1,A
            MOV  A,R0
            MOV  @R1,A
            DJNZ  R5,INW1
            SETB INPFIR
            SJMP INW
INW3:       MOV  DISBUF+0,R0
            MOV  DISBUF+1,#20H
            MOV  DISBUF+2,#20H
            MOV  DISBUF+3,#20H
            MOV  DISBUF+4,#20H
            MOV  DISBUF+5,#20H
            LCALL WLINE31
            MOV  A,#4
            LCALL GOTOXY
            CLR  INPFIR
            LJMP INW1
INW5:       CJNE  A,#28H,INW1
            MOV R2,DISBUF+0
            MOV  R3,DISBUF+1
            LCALL ATOH                  ;EXIT
            MOV  HEXBUF+0,A
            MOV R2,DISBUF+2
            MOV  R3,DISBUF+3
            LCALL ATOH                  ;EXIT
            MOV  HEXBUF+1,A
            CLR  INPBFG
            RET
```

```
;******GET1 SUB*******
;GET HEX ON DIGIT 4,5
;IN = A KEY CODE 0-FH
;OUT= DISBUF
;    =GETOKF(2 DIGIT COMPLETE)
;REG= A,R2,DPTR
GET1:       CLR  GETOKF
            PUSH ACC
            CLR  C
            SUBB A,#10H          ;COMPARE
            POP  ACC
            JC   GET11
            RET

GET11:      MOV  R2,A
            LCALL HTOAS
            MOV  R2,A
            MOV  A,DISBUF+6
            ANL  A,#0DFH          ;AND DOT
            CJNE A,#0H,GET15
            MOV  A,R2
            MOV  DISBUF+6,A       ;LAST DIGIT
            MOV  A,#41H
            LCALL GOTOXY
            MOV  A,DISBUF+6
            MOV  R0,A
            LCALL   WRITECHAR
            SETB GETOKF
            LCALL LDELAY
            RET
GET15:      MOV  DISBUF+5,R2      ;FIRST DIGIT
            MOV  A,R2
            MOV  DISBUF+6,#20H
            LCALL WLINE6
            RET
WLINE6:             MOV     A,#40H
                    PUSH    DPH
                    PUSH    DPL
                    LCALL   GOTOXY
                    MOV     B,#2
                    POP     DPL
                    POP     DPH
                    MOV     R1,#DISBUF+5
                    LCALL   WRITE1
                    RET
;*************************************************************
DPDEC:              XCH     A,DPL
                    JNZ     $+4
                    DEC     DPH
                    DEC     A
                    XCH     A,DPL
                    RET
```

```
;*************************************************************
;WLINE1 IS WRITE FIRST LINE CHARACTER (16 CHARACTERS VECTOR BY DPTR
;WRITTEN AND INCREASE IN "WRITE" ROUTINE
WLINE1:         MOV     A,#00H
                PUSH    DPH
                PUSH    DPL
                LCALL   GOTOXY
                MOV     B,#8
                POP     DPL
                POP     DPH
                LCALL   WRITE
                RET
WLINE2:         MOV     A,#40H
                PUSH    DPH
                PUSH    DPL
                LCALL   GOTOXY
                MOV     B,#8
                POP     DPL
                POP     DPH
                LCALL   WRITE
                RET
;*************************************************************
WLINE3:         MOV     A,#00H
                PUSH    DPH
                PUSH    DPL
                LCALL   GOTOXY
                MOV     B,#3
                POP     DPL
                POP     DPH
                LCALL   WRITE
                RET

;*************************************************************
WLINE31:        MOV     A,#03H
                PUSH    DPH
                PUSH    DPL
                LCALL   GOTOXY
                MOV     B,#5
                POP     DPL
                POP     DPH
                MOV     R1,#DISBUF+0
                LCALL   WRITE1
                RET
;*************************************************************
WLINE4:         MOV     A,#03H
                PUSH    DPH
                PUSH    DPL
                LCALL   GOTOXY
                MOV     B,#5
                POP     DPL
                POP     DPH
                MOV     R1,#DISBUF+0
                LCALL   WRITE1
```

```
                        MOV     A,#40H
                        PUSH    DPH
                        PUSH    DPL
                        LCALL   GOTOXY
                        MOV     B,#2
                        POP     DPL
                        POP     DPH
                        MOV     R1,#DISBUF+5
                        LCALL   WRITE1
                        RET
;*****************************************************************
WLINE5:                 MOV     A,#06H
                        PUSH    DPH
                        PUSH    DPL
                        LCALL   GOTOXY
                        MOV     B,#2
                        POP     DPL
                        POP     DPH
                        MOV     R1,#DISBUF+3
                        LCALL   WRITE1
                        RET
;*****************************************************************

;THIS ROUTINE IS CALLED FOR WRITING CHARACTER FROM CG-RAM TO DD-RAM
WRITE:                  MOV     A,#0
                        MOVC    A,@A+DPTR
                        MOV     R0,A            ;R0 KEEP TO WRITTEN
                        PUSH    DPH
                        PUSH    DPL
                        LCALL   WRITECHAR
                        POP     DPL
                        POP     DPH
                        INC     DPTR
                        DJNZ    B,WRITE
                        RET
;*****************************************************************
;THIS ROUTINE IS CALLED FOR WRITING CHARACTER FROM CG-RAM TO DD-RAM
WRITE1:                 MOV     A,@R1
                        MOV     R0,A            ;WRITTEN KEEP IN R0
                        PUSH    DPH
                        PUSH    DPL
                        LCALL   WRITECHAR
                        POP     DPL
                        POP     DPH
                        INC     R1
                        DJNZ    B,WRITE1
                        RET
;*****************************************************************
;THIS ROUTINE IS USED FOR WRITE EXTERNAL RAM DATA TO LCD DISPLAY
WRITE_EXT:              MOVX    A,@DPTR
                        MOV     00H,A           ; WRITTEN KEEP IN R0
                        PUSH    DPH
                        PUSH    DPL
```

```
                        LCALL   WRITECHAR
                        POP     DPL
                        POP     DPH
                        INC     DPTR
                        DJNZ    B,WRITE_EXT
                        RET
;*************************************************************************
;WRITECHAR IS USED FOR DISPLAY ONE CHARACTER FROM CG-RAM TO DD-RAM
;THE CHARACTER TO BE WRITTEN IS KEPT IN REG.R0
WRITECHAR:              PUSH    00H
                        PUSH    ACC
                        MOV     A,#00000001B    ;SET RS=1
                        MOV     DPTR,#PORT_B
                        MOVX    A,@DPTR
                        SETB    ACC.0
                        MOVX    @DPTR,A
                        MOV     DPTR,#0F800H
                        MOV     A,R0
                        MOVX    @DPTR,A
                        MOV     A,#1
                        LCALL   ENABLEPULSE
                        LCALL   DELAY
                        POP     ACC
                        POP     00H
                        RET
;*************************************************************
;REG.A INITIAL DD-RAM ADDRESS
GOTOXY:                 SETB    ACC.7
                        MOV     DPTR,#0F800H
                        MOVX    @DPTR,A
                        MOV     DPTR,#PORT_B
                        MOV     A,#0
                        MOVX    A,@DPTR
                        ANL     A,#11111000B
                        MOVX    @DPTR,A
                        LCALL   ENABLEPULSE
                        LCALL   DELAY
                        RET

;*************************************************************
;INITIAL 8255
CTRL8255                EQU     0F803H
PORT_A                  EQU     0F800H
PORT_B                  EQU     0F801H
PORT_C                  EQU     0F802H

KEY_LCD_8255:           MOV     A,#89H
                        MOV     DPTR,#CTRL8255
                        MOVX    @DPTR,A
                        MOV     DPTR,#PORT_B
                        MOV     A,#08H
                        MOVX    @DPTR,A
                        RET
```

```
;*****************************************************************
INITIAL_LCD:            MOV     A,#11111000B  ;SET RS=0 , R/W=0 , E=
                        MOV     DPTR,#PORT_B
                        MOVX    @DPTR,A
                        LCALL   FUNC_SET
                        LCALL   LDELAY
                        LCALL   DISP_CTRL
                        LCALL   LDELAY
                        LCALL   ENTRY_SET
                        LCALL   LDELAY
                        LCALL   CLEARSCREEN
                        LCALL   LDELAY
                        RET
;*****************************************************************
;FUNCTION SET : DL=1(8 BIT DATA) ,N=1(2 LINES DISPLAY),F=0 (5x7 DOT)
FUNC_SET:               MOV     A,#00111000B
                        MOV     DPTR,#PORT_A
                        MOVX    @DPTR,A
                        LCALL   ENABLEPULSE
                        LCALL   DELAY
                        RET
;*****************************************************************
;DISPLAY ON-OFF CONTROL : DISPLAY=ON , CURSOR=OFF
DISP_CTRL:              MOV     A,#00001100B
                        MOV     DPTR,#PORT_A
                        MOVX    @DPTR,A
                        LCALL   ENABLEPULSE
                        LCALL   DELAY
                        RET
;*****************************************************************
;ENTRY MODE SET : CURSOR SHIFT RIGHT , INCREASE DD-RAM ADDRESS
ENTRY_SET:              MOV     A,#00000110B
                        MOV     DPTR,#PORT_A
                        MOVX    @DPTR,A
                        LCALL   ENABLEPULSE
                        LCALL   DELAY
                        RET
;*****************************************************************
CLEARSCREEN:            LCALL   LDELAY
                        MOV     DPTR,#0F801H
                        MOVX    A,@DPTR
                        ANL     A,#11111000B
                        MOVX    @DPTR,A
                        MOV     A,#00000001B
                        MOV     DPTR,#PORT_A
                        MOVX    @DPTR,A
                        LCALL   ENABLEPULSE
                        LCALL   LDELAY
                        RET
;*****************************************************************
RETURNHOME:             MOV     A,#00000010B
                        MOV     DPTR,#PORT_A
                        MOVX    @DPTR,A
```

```
                        MOV     A,#0
                        LCALL   ENABLEPULSE
                        LCALL   DELAY
                        RET
;*************************************************************
;ENABLEPULSE ROUTINE GENERATE ONE PULSE SIGNAL THAT FALLING
 EDGE ENABLE LCD-MODULE
;GET INSTRUCTION DATA FROM PORT 1 OF 8052 CPU
ENABLEPULSE:            PUSH    ACC
                        PUSH    B
                        PUSH    DPH
                        PUSH    DPL
                        MOV     DPTR,#0F801H
                        MOVX    A,@DPTR
                        ORL     A,#00000100B
                        MOVX    @DPTR,A
                        MOV     B,#00H
ENABLELOOP:             DJNZ    B,ENABLELOOP
                        ANL     A,#11111011B
                        MOVX    @DPTR,A
                        POP     DPL
                        POP     DPH
                        POP     B
                        POP     ACC
                        RET
;*************************************************************
;DELAY 740T STATE AT 11.0592 MHz IS ABOUT 61.6 MICROSECONDS
DELAY:                  PUSH    B
                        MOV     B,#100
DELAYLOOP:              DJNZ    B,DELAYLOOP
                        POP     B
                        RET

;*************************************************************
;KDELAY IS PURPOSE FOR DELAY 1.026 mSEC AT 11.0592 MHz
KDELAY:                 PUSH    B
                        MOV     B,#0FFH
KLOOP:                  NOP
                        NOP
                        NOP
                        NOP
                        DJNZ    B,KLOOP
                        POP     B
                        RET
;*************************************************************
;LDELAY IS THE LONGEST DELAY LOOP IN THIS PROGRAM
LDELAY:                 MOV     06H,#0FFH
LDE1:                   MOV     07H,#0FFH
LDE2:                   DJNZ    07H,LDE2
                        DJNZ    06H,LDE1
                        RET
;*************************************************************
SCAN:                   MOV     R1,#0
```

```
                MOV     R7,#0
                MOV     R0,#9
                MOV     DPTR,#0F801H
                MOVX    A,@DPTR
                ANL     A,#0FH
                MOV     R7,A
                MOV     A,0
SCAN1:          MOV     DPTR,#0F801H
                MOVX    @DPTR,A
                MOV     DPTR,#0F802H
                MOVX    A,@DPTR
                MOV     R3,A
                ORL     A,#07H
                CJNE    A,#0FFH,ENCODE
                MOV     A,R7
                ADD     A,#10H
                MOV     R7,A
                MOV     A,R1
                ADD     A,#5
                MOV     R1,A
                MOV     A,R7
                DJNZ    R0,SCAN1
                SJMP    SCAN
                RET
ENCODE:         MOV     DPTR,#0F802H
                MOVX    A,@DPTR
                MOV     B,#0F0H
EN:             DJNZ    B,EN
                ORL     A,#07H
                CJNE    A,#0FFH,ENCODE
                LCALL   LDELAY
ENCODE2:        MOV     A,R3
ENCODE23:       RLC     A
                JNC     ENCODE1
                INC     R1
                SJMP    ENCODE23
ENCODE1:        MOV     A,R1
                CJNE    A,#10H,$+6
                LJMP    DEBUG
                CJNE    A,#0,$+7
                MOV     A,#4
                SJMP    RA
                CJNE    A,#1,$+7
                MOV     A,#3
                SJMP    RA
                CJNE    A,#2,$+7
                MOV     A,#2
                SJMP    RA
                CJNE    A,#3,$+7
                MOV     A,#1
                SJMP    RA
                CJNE    A,#4,$+7
                MOV     A,#0
```

```
                    SJMP    RA
                    CJNE    A,#0EH,$+7
                    MOV     A,#0AH
                    SJMP    RA
                    CJNE    A,#13H,$+7
                    MOV     A,#0BH
                    SJMP    RA
                    CJNE    A,#18H,$+7
                    MOV     A,#0CH
                    SJMP    RA
                    CJNE    A,#1DH,$+7
                    MOV     A,#0DH
                    SJMP    RA
                    CJNE    A,#22H,$+7
                    MOV     A,#0EH
                    SJMP    RA
                    CJNE    A,#27H,$+7
                    MOV     A,#0FH
                    SJMP    RA
                    CJNE    A,#20H,$+6
                    LJMP    INTR
                    CJNE    A,#26H,$+6
                    LJMP    DMEM
                    CJNE    A,#2CH,$+6
                    LJMP    GO

RA:                 RET
TABLE:              DB      " TRAININ"
TABLE1:             DB      "G BOARD "
TABLE2:             DB      "<S>:"
TABLE3:             DB      "<M>:"
TABLE4:             DB      "<R>:"
TABLE5:             DB      "<G>:"
TABLE6:             DB      "<F>:"


;*****HTOA SUB*****60
;CONVERT HEX TO ASCII
;IN = A
;OUT= R2,R3
;REG= A,R2,R3
HTOA:       PUSH ACC
            SWAP A
            LCALL HTOAS
            MOV  R2,A
            POP  ACC
            LCALL HTOAS
            MOV  R3,A
            RET
HTOAS:      ANL  A,#0FH
            CJNE A,#0AH,$+3
            JNC  HTOAS1
```

```
                ORL   A,#30H
                RET
HTOAS1:         SUBB  A,#9
                ORL   A,#40H
                RET
;*****ATOH SUB*****61
;ASCII TO HEX CONVERT
;IN= R2,R3 30H,41H
;OUT=A      0AH
;REG=A,R2
ATOH:           MOV   A,R2
                LCALL ATOHS
                SWAP  A
                MOV   R2,A
                MOV   A,R3
                LCALL ATOHS
                ORL   A,R2
                RET
ATOHS:          CJNE  A,#'A',$+3
                JC    ATOHS1
                ADD   A,#9
ATOHS1:         ANL   A,#0FH
                RET
;*****HTOD SUB*****62
;HEX TO DECIMAL
;IN = DPTR
;OUT= R1,R2,R3
;REG= A,R0,R1,R2,R3,R4,R5,DPTR
HTOD:           CLR   A
                MOV   R1,A
                MOV   R2,A
                MOV   R3,A
                MOV   R4,#16            ;SHIFT 16 BIT
HTOD1:          MOV   A,DPL
                RLC   A
                MOV   DPL,A
                MOV   A,DPH
                RLC   A
                MOV   DPH,A
                MOV   R5,#3             ;ADD DECIMAL
                MOV   R0,#3             ;INDEX TO R3
HTOD2:          MOV   A,@R0
                ADDC  A,ACC
                DA    A
                MOV   @R0,A
                DEC   R0
                DJNZ  R5,HTOD2
                DJNZ  R4,HTOD1
                RET
;*****DTOH  SUB******63
;DECIMAL TO HEX
;IN = R1,R2,R3
;OUT= DPTR
```

```
;REG= A,R0,R1,R2,R3,R4,R5,DPTR

DTOH:       MOV   R4,#16
DTOH1:      MOV   R5,#3             ;SHIFT & SUB
            MOV   R0,#1             ;INDEX TO R1
            CLR   C
DTOH2:      MOV   A,@R0
            RRC   A
            PUSH  PSW               ;-[
            JNB   ACC.7,DTOH3
            CLR   C
            SUBB  A,#30H
DTOH3:      JNB   ACC.3,DTOH4
            CLR   C
            SUBB  A,#03H
DTOH4:      MOV   @R0,A
            INC   R0
            POP   PSW               ;-]
            DJNZ  R5,DTOH2
            MOV   A,DPH
            RRC   A
            MOV   DPH,A
            MOV   A,DPL
            RRC   A
            MOV   DPL,A
            DJNZ  R4,DTOH1
            RET
;**********DPSUB SUB********1
;DPTR=DPTR-R2,R3
;IN=DPTR,R2,R3
;OUT=DPTR
;REG=A,DPTR

DPSUB:      CLR   C
            MOV   A,DPL
            SUBB  A,R3
            MOV   DPL,A
            MOV   A,DPH
            SUBB  A,R2
            MOV   DPH,A
            RET
;*********DPCOM SUB*********14
;COMPARE WORD(16BIT) DPTR<>R2,R3
;IN=DPTR,R2,R3
;OUT=CY,A
;REG=A

DPCOM:      PUSH  DPL
            CLR   C
            MOV   A,DPL
            SUBB  A,R3
            MOV   DPL,A
            MOV   A,DPH
```

```
        SUBB  A,R2
        ORL   A,DPL
        POP   DPL
        RET
SFRDF:  MOV   VTSP,#07H
        MOV   VTTCON,#40H
        MOV   VTTMOD,#20H
        MOV   VTTH1,#0FDH
        MOV   VTP1,#0FFH
        MOV   VTSCON,#52H
        MOV   VTIE,#0E4H
        MOV   VTIP,#0E0H
        RET
              END
```

กิตติกรรมประกาศ

การจัดทำโครงงานในครั้งนี้สำเร็จลุล่วงลงไปได้ก็เพราะได้รับทั้งความร่วมมือและการช่วยเหลือจากบุคคลที่เป็นเพื่อน เป็นอาจารย์ คนรู้จัก และบริษัทที่ประกอบการเกี่ยวกับระบบทางคอมพิวเตอร์หลายแห่ง หากจะให้ระบุเป็นรายๆก็อาจจะเกิดการตกหล่นไปบ้าง เอาเป็นว่าใครก็ตามที่เคยได้ให้ความช่วยเหลือกลุ่มโครงงาน MCS-51 TRAINING BOARD มากบ้างน้อยบ้าง ก็ต้องขอขอบคุณในความเอื้อเฟื้อมา ณ โอกาสนี้ เสียทีเดียวเลย

บรรณานุกรม

Signatics

Microcontroller

Users' Guide

Philip Co.,Ltd

Z-80 Assembly Language Programming

Lance A. Leventhal

Z-80 Application

James W. Coffron

Sybex Inc.

JAZZ-31 Single Board Microcontroller

Monitor Program V 1.1

เอกสารประกอบการเรียนวิชาไมโครโปรเซสเซอร์

อ. พิพัฒน์ เลาหสงคราม

ทฤษฎีและการประยุกต์ใช้ไมโครโปรเซสเซอร์ Z-80

ยืน ภู่สุวรรณ

บริษัท ซีเอ็ดยูเคชั่น จำกัด