ปริญญานิพนธ์ปีการศึกษา 2533

ภาควิชา เทคนิคอุตสาหกรรม

คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องควบคุมระบบในรถยนต์

ผู้จัดทำ

1. นายวิทยา จำรัสฉาย
2. นายวัชร ชื่นบุญ

.........................อาจารย์ที่ปรึกษา

(ผศ. นิกร สุขุตมตันติ)

028792

13 มี.ค. 2534

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ปริญญานิพนธ์ ปีการศึกษา ๒๕๓๓

เรื่อง Auto mobile controller

ผู้จัดทำ

๑.นาย วัชร ชื่นบุญ

๒.นาย วิทยา จำรัสฉาย

....อาจารย์ที่ปรึกษา

(ผศ. นิกร สุขุตมตันติ)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

028792

Auto mobile controller

นายวัชร ชื่นบุญ

นายวิทยา จำรัสฉาย

ผศ. นิกร สุขุตมตันติ อาจารย์ที่ปรึกษา

ปีการศึกษา ๒๕๓๓

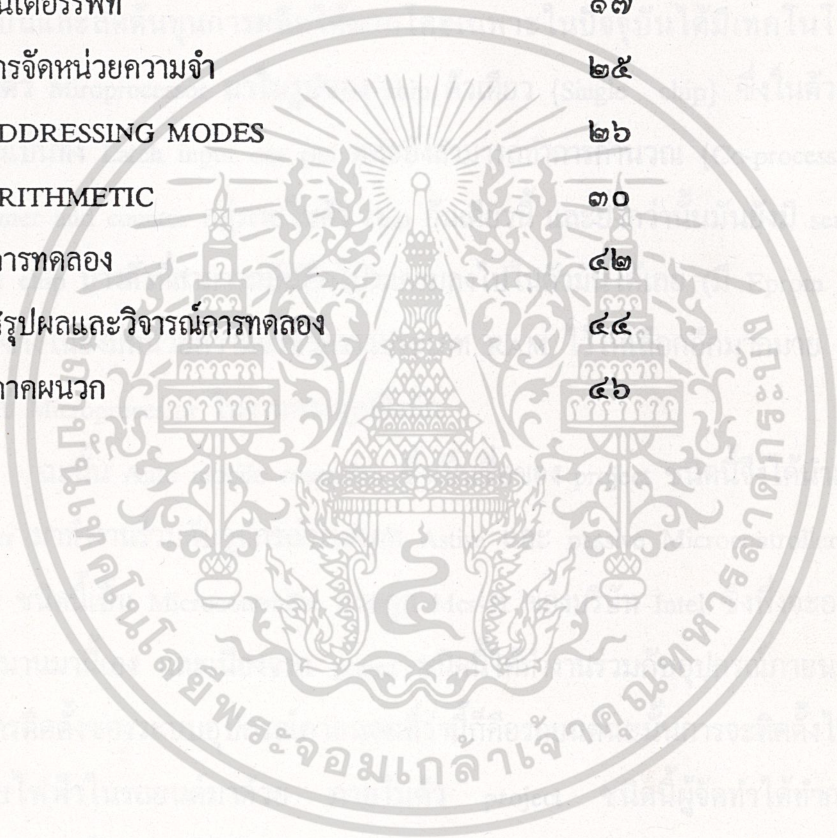
บทคัดย่อ

วิทยานิพนธ์ฉบับนี้ เรียบเรียงขึ้นจากผลงานการออกแบบและทดสอบอุปกรณ์ทางพวก Sensor ควบคู่กับการนำเอา Microcontroller มาประยุคต์ให้ใช้งานร่วมกับอุปกรณ์ประเภท Active และ Passive เพราะว่า Micrecontroller สามารถนำเอาสัญญาณจากตัว Sensor มาทำงาน ประมวลพลได้เกือบ ๑๐๐% เลยที่เดียว เพราะฉะนั้นเมื่อนำเอาวงจร Sensor และผลจากการทำงานร่วมกับ Machanic แล้วนำเอามา intu phau ร่วมกับตัว Microcontroller ซึ่งอาศัย เทคนิคบางประการนี้สมารถใช้ระบบ ทำงานได้ตามเป้าหมาย จากผลของการทำงานอันนี้จะเห็นว่าหัวโจทย์ คือตัว Micrecontroller

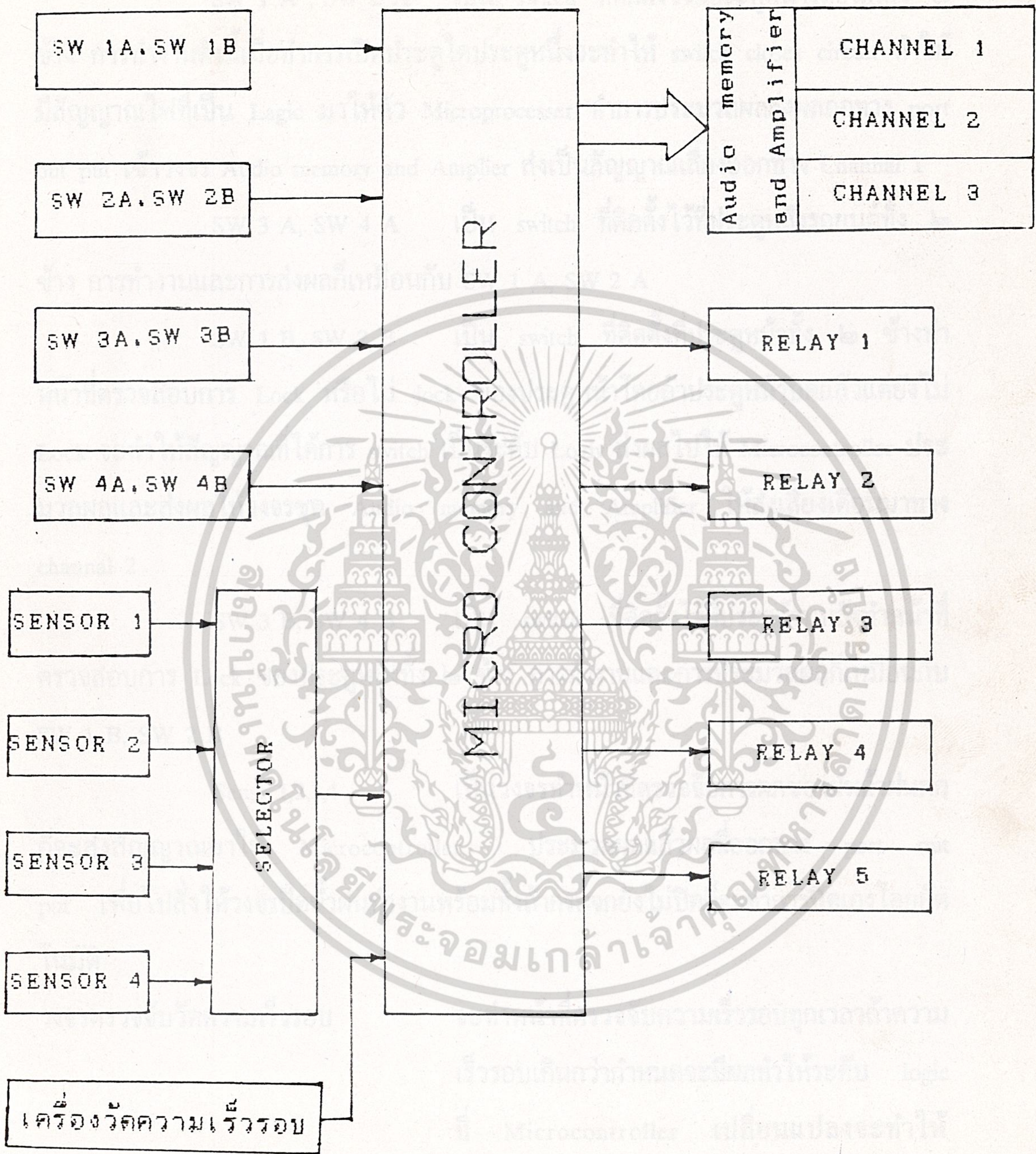
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ

๑. บทนำ	๑
๒. ลักษณะโครงสร้างของ Mes-51	๖
๓. การจัดหน่วยความจำของ ๘๐๓๑	๗
๔. โครงสร้างและการทำงานของพอร์ท	๑๒
๕. การติดต่อกับหน่วยความจำภายนอก	๑๖
๖. อินเตอร์รัพท์	๑๗
๗. การจัดหน่วยความจำ	๒๕
๘. ADDRESSING MODES	๒๖
๙. ARITHMETIC	๓๐
๑๐.การทดลอง	๔๒
๑๑.สรุปผลและวิจารณ์การทดลอง	๔๔
๑๒.ภาคผนวก	๔๖



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



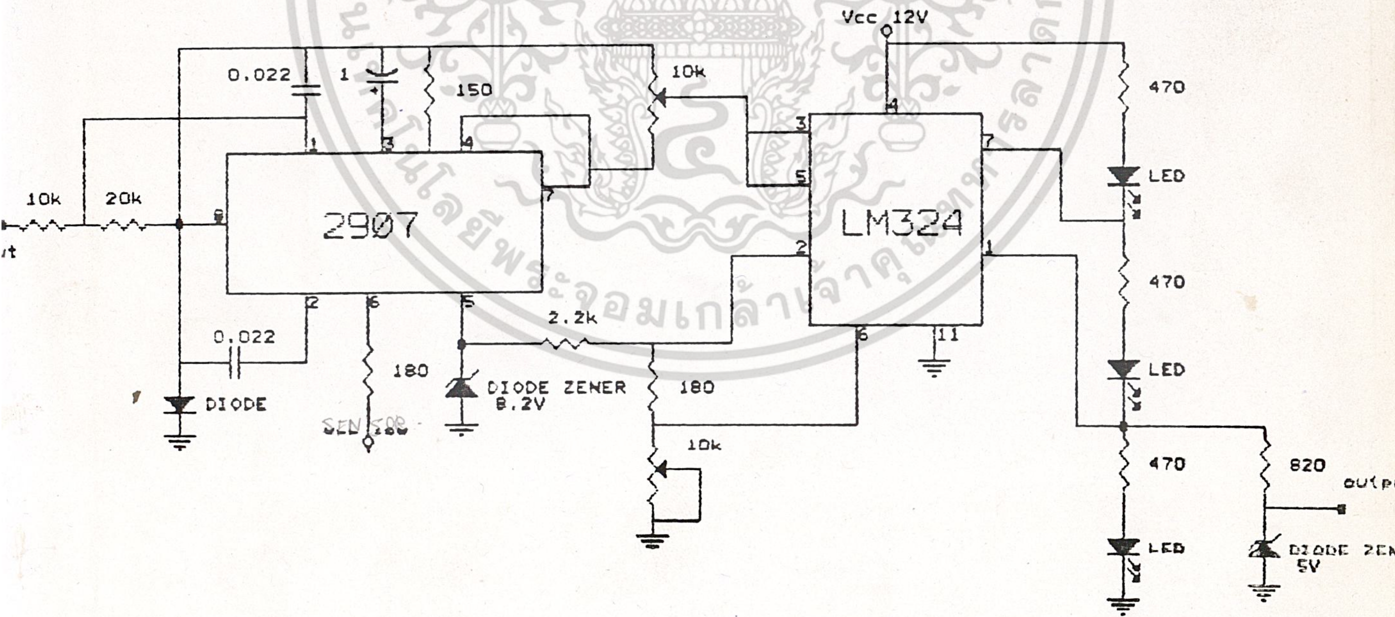
SENSOR 1 : เป็นแผ่นทองแดง ที่ใช้ตรวจจับความชื้น โดยนำสัญญาณมาขยาย

SENSOR 2 : เป็นแผ่นทองแดง ที่ใช้ตรวจจับความชื้นโดยได้สัญญาณเป็น DC

SENSOR 3 : 〃 〃 〃 〃

SENSOR 4 : 〃 〃 〃 〃

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ลักษณะโครงสร้างของ Mcs-51

MCS - 51

ไมโครโปรเซสเซอร์ตระกูล MCS - 51 เป็นไมโครโปรเซสเซอร์ แบบซิงเกิลชิพ เพราะภายในชิพประกอบด้วยหน่วยความจำ ส่วนการคำนวณ ส่วนสร้างสัญญาณนาฬิกา พอร์ทแบบขนาน วงจรนับเวลาและตั้งเวลา พอร์ทแบบอนุกรมทำให้มีการให้งานอย่างกว้างขวาง เพราะในการออกแบบระบบจะไม่ต้องใช้อุปกรณ์ภายนอกมาก จึงประหยัดทั้งพลังงานที่ใช้และการลงทุนในระบบ ไมโครคอมพิวเตอร์ตระกูล MCS - 51 มีข้อดีเหนือกว่าไมโครโปรเซสเซอร์ตัวอื่น ๆ ที่มีการใช้งานในปัจจุบันมาก เพราะมีหน่วยความจำสำหรับใช้เขียนโปรแกรมได้ถึง ๖๔ กิโลไบท์ และมีหน่วยความจำสำหรับการใช้เก็บข้อมูลแบบออกมาอีก ๖๔ กิโลไบท์ และมีหน่วยความจำสำหรับการใช้เก็บข้อมูลแยกออกมาอีก ๖๔ กิโลไบท์ หน่วยความจำภายในแบบแรนด้อมแอกเซส (RAM) ขนาด ๒๕๖ ไบท์ จะมีบางส่วนทีโปรแกรมจะสามารถติดต่อได้ทีละบิท และแบบไบท์พอร์ทขนานภายในสามารถใช้งานได้ทั้งแบบอินพุท และเอาท์พุท และ พอร์ท ๓ สามารถใช้งานในฟังก์ชั่นอื่นได้ วงจรนับเวลาและตั้งเวลาภายใน ๒ ชุด ขนาด ๑๖ บิท (TIMER O TIMER 1) สามารถโปรแกรมเพื่อการใช้งานแยกจากกันได้อย่างอิสระ สามารถใช้โปรแกรมกำหนดอัตราการส่งข้อมูล (BAUD RATE) ของพอร์ทอนุกรม ซึ่งเป็นฟังช์ชั่นของพอร์ท ๓ สามารถเลือกให้ทำงานจากโปรแกรมที่อยู่ภายในชิพหรือภายนอกชิพ โดยการต่อขาหนึ่งเข้ากับ ลอจิก ๐ หรือ ลอจิก ๑ อินเทอร์รัพท์ภายนอก ๒ อินเทอร์รัพท์ และภายใน ๓ อินเทอร์รัพท์ ทำให้ใช้งานควบคุมได้อย่างมีประสิทธิภาพ

ลักษณะของ ๘๐๓๑

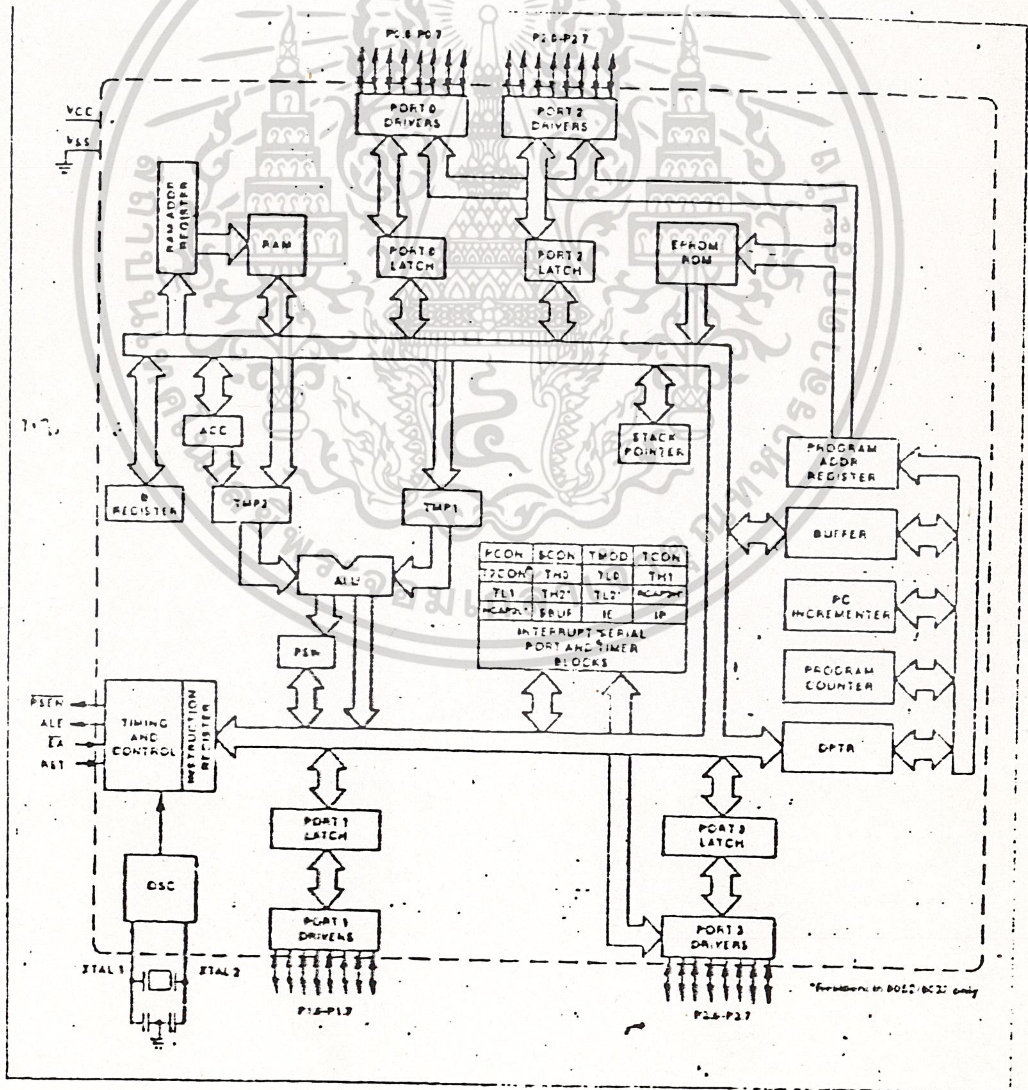
- เป็น CPU แบบ ๘ บิท
- มีวงจรออสซิลเลเตอร์และ CLOCK อยู่ในตัว
- มีขาสัญญาณเข้าและออก (I/O) ๓๒ ขา
- แยกหน่วยความจำของข้อมูลได้ ๖๔ K และหน่วยความจำของโปรกรมอีก ๖๔ K
- มี TIMER และCOUNTER แบบ ๑๖ บิท ถึง ๒ ตัว
- สัญญาณอินเทอร์รัพท์ ๕ แบบ ซึ่งแบ่งลำดับความสำคัญออกเป็น ๒ระดับ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- การทำงานแบบ FULL DUPLEX ในขณะส่งข้อมูลแบบอนุกรม
- มีการประมูลผลแบบบูลีน (AND, OR, XOR) ฯลฯ

การจัดหน่วยความจำของ ๘๐๓๑

๘๐๓๑ แบ่งหน่วยความจำสำหรับโปรแกรมและข้อมูลอย่างละ ๖๔ K ในส่วนของหน่วยความจำของข้อมูลจะเป็น RAM ซึ่งเป็นหน่วยความจำภายนอก ๖๔ K และยังมีหน่วยความจำประเภทแรมอีก ๑๒๘ ไบท์ ซึ่งอยู่ในตัวของ ๘๐๓๑ และบน ๘๐๓๑ ยังประกอบด้วยรีจิสเตอร์ที่ทำหน้าที่พิเศษ 'SFRS' (SPECIAL FUNCTION REGISTOR) ซึ่งแสดงในตารางที่ ๑



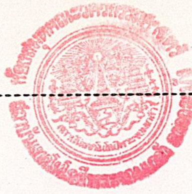
รูป ๑ โครงสร้างของ ๘๐๓๑

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ ๑

SYMBOL	NAME	ADDRESS
* ACC	ACCUMULATOR	OEOH
* B	B REGISTOR	OROH
* PSW	PROGRAM STATUS WORD	ODOH
SP	STACK POINTER	81H
DPIR	DATA POINTER {CONSIST OF DPH,PPL}	83H
	DPL	82H
* PO	PORT 0	80H
* P1	PORT 1	90H
* P2	PORT 2	OAOH
* P3	PORT 3	OBOH
* 1P	INTERUPT PIORITY CONTROL	OB8H
* 1E	INTERUPT ENABLE CONTROL	OA8H
TMOD	TIMER/COUNTER MODE CONTROL	89H
+ *T2CON	TIMER/COUNTER CONTROL	OC8H
TCON	TIMER/COUNTER 2 CONTROL	88H
THO	" / " 0 {HIGH BYTE}	8CH
TLO	" / " 0 {LOW BYTE}	8CH
TH1	" / " 1 {HIGH BYTE}	8DH
TL1	" / " 1 {LOW BYTE}	8BG
+ TH2	" / " 2 {HIGH BYTE}	0CDH
+ TL2	" / " 2 {LOW BYTE}	0CCH
+ RCAP2H	" / " 2 CAPTURE REG {H1}	0CBH
+ RCAP2L	" / " 2 CAPTURE REG {LOW}	0CAH
* SCON	SERIAL CONTROL	98H
SBUF	SERIAL DATA BUFF	99H
PCON	POWER CONTROL	87H

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



หมายเหตุ * SFRs ที่มีเครื่องหมายดอกจันทร์อยู่ข้างหน้า หมายความว่า สามารถเข้าถึงได้ โดยการอ่านแอดเดรสแบบไบท์ และแบบบิท คือสามารถจะเข้าถึงบิทใดบิทหนึ่งใน SFRs ได้โดยตรง

+ SFRs ที่มีเครื่องหมายบวกแสดงว่ามีอยู่ใน ๘๐๓๒/๘๐๕๒ เท่านั้น

รายละเอียดของรีจิตสเตอร์ที่ทำหน้าที่พิเศษ

ACCUMULATOR

ACC เป็นเอคคิวมูเลเตอร์หรือรีจิสเตอร์ A เหมือนกับ Z๘๐

B REGISTOR

รีจิสเตอร์ B ใช้ในคำสั่งคูณและหาร ส่วนในคำสั่งอื่นสามารถใช้เหมือนกันรีจิสเตอร์ทั่ว ๆ ในการการพักข้อมูล

PROGRAM STATUS WORD

RSW ประกอบด้วยรายละเอียดดังในรูป ๒

PSW: PROGRAM STATUS WORD

{MSB}							{LSB}
CY	AC	FO	RS1	RS0	OV	-	P

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้
028792

SYMBOL	POSITION	NAME AND SIGNIRICANCE
CY	PSW.7	CARRY FLAG
AC	PSW.6	AUXILLARY CARRY FLAG {FOR BCD OPERATION}
FO	PSW.5	FLAG 0 {FOR GENERAL PURPOSES}
RS1	PSW.4	REGISTOR BANK SELECT CONTROL BITS 1 & 0 SET/CLEAREL BY SOLFWARE TO DETERMINE
RSO	PSW.3	WORKING REGISTOR BANK {SEE NOTE}
OV	PSW.2	OVERFLOW FLAG
-	PSW.1	{RESERVED}
P	PSW.0	PARIY FLAG. SET/CLEARED BY HARDWARE EACH INSTRUCTION TO IBDICATE AN ODD/EVEN NUNBER OF"ONE"BITS IN THE ACCUMULATOR., IE. EVEN PAFITY.

NOTE- THE CONSISTS OF {RS1, RSO}ENABLE THE WORKING REGISTOR BANKS AS FOLLOWS.

{0.0} - BANK 0 {00H-07H}

{0.1} - BANK 1 {08H-0FH}

{1.0} - BANK 2 {10H-17H}

{1.1} - BANK 3 {18H-1FH}

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น มีทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

STACK POINTER

SP เป็นรีจิสเตอร์ขนาด ๘ บิท เมื่อใช้คำสั่ง PUSH และ CALL SP จะเพิ่มขึ้นก่อนที่จะเก็บข้อมูลหรือแอดเดรส ซึ่ง STACK จะอยู่ที่ไหนก็ได้ในหน่วยความจำ {RAM} บน ๘๐๓๑ ภายหลังการ RESET แสตกจะมาอยู่ที่ ๐๗H แสตกจะเริ่มที่ ๘๐H

DATA POINTER

DPTR ประกอบด้วย DPH และ DPL ซึ่งจะรวมกันเป็นรีจิสเตอร์ขนาด ๑๖ บิท การเข้าถึง DPTR สามารถทำได้ทั้งแบบ ๑๖ บิท และ แบบ ๘ บิท หน้าที่ของ DPTR ถ้าใช้แบบ ๑๖ บิท จะทำหน้าที่เป็นตัวชี้ข้อมูลที่อยู่ในหน่วยความจำภายนอก

PORT 0 - 3

พอร์ต ๐, พอร์ต ๑, พอร์ต ๒, และพอร์ต ๓ เป็น SFR ตัวหนึ่งที่สามารถแลตช์ข้อมูลได้สามารถใช้เป็นอินพุทพอร์ต และเอ๊าท์พุทพอร์ตได้ทั้ง ๓ พอร์ต

SERIAL DATA BUFFER

แบ่งเป็นรีจิสเตอร์บัฟเฟอร์ในทางส่งและบัฟเฟอร์ในทางรับ เมื่อมีการส่งข้อมูลภายในให้ SBUF ข้อมูลจะถูกส่งมาที่ บัฟเฟอร์ในทางส่ง ที่ซึ่งจะใช้ทำการส่งข้อมูลอนุกรม (การส่งข้อมูลให้ SBUF จะเป็นการเริ่มต้นการส่งด้วย) และถ้าอ่านข้อมูลจาก SBUF ข้อมูลจะถูกอ่านจากบัฟเฟอร์ในทางรับ

TIMER REGISTORS

รีจิสเตอร์คู่ (THO, TLO) และ (TH 1, TL1) เป็นตัวจับเวลาและตัวนับขนาด ๑๖ บิท จะกล่าวรายละเอียดภายหลัง

CONTROL REGISTORS

รีจิสเตอร์หน้าที่พิเศษ TP, IE, TMOD, TCON, T3CON, SCON และ PCON ประกอบด้วยบิทควบคุมแบะบิทสถานะสำหรับระบบการอินเตอร์รัพท์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

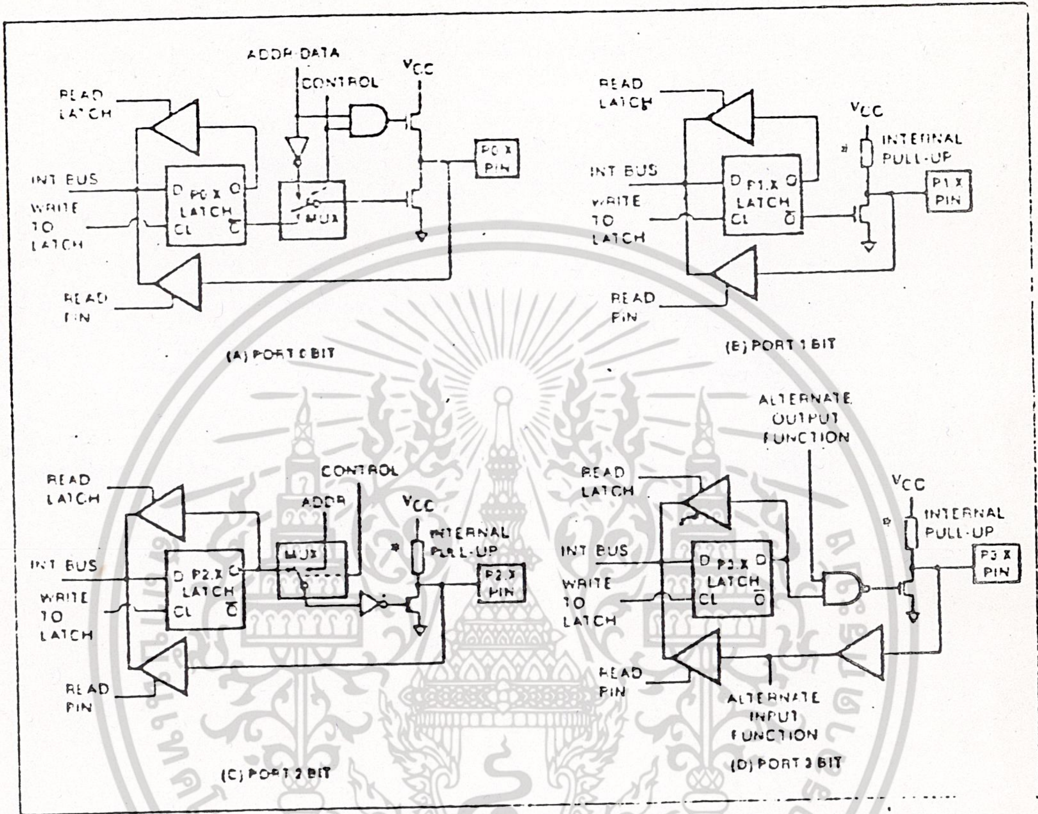
โครงสร้างและการทำงานของพอร์ต

๘๐๓๑ มี I/O พอร์ต อยู่ ๔ พอร์ต โดยแต่ละพอร์ตจะเป็นพอร์ตแบบ ๒ ทิศทาง มีการแลตช์ข้อมูลได้ (SFR.PO-P3) รวมทั้งมีวงจรขับทางเอ๊าท์พุท และบัฟเฟอร์ทางด้านอินพุท

พอร์ต ๐ และพอร์ต ๒ ใช้สำหรับติดต่อกับหน่วยความจำภายนอก ในการใช้ ๘๐๓๑ ติดต่อกับหน่วยความจำภายนอก พอร์ต ๐ จะให้เอ๊าท์พุทเป็น LOW BYTE ของแอดเดรสของหน่วยความจำภายนอกและจะทำการ MULTIPLEX กับข้อมูลที่จะเขียนหรืออ่าน สรุปคือ PO จะเป็นทั้ง ADDRESS และ DATA ส่วนพอร์ต ๒ จะให้แอดเดรสไบต์สูง (MSB) ของหน่วยความจำภายนอก

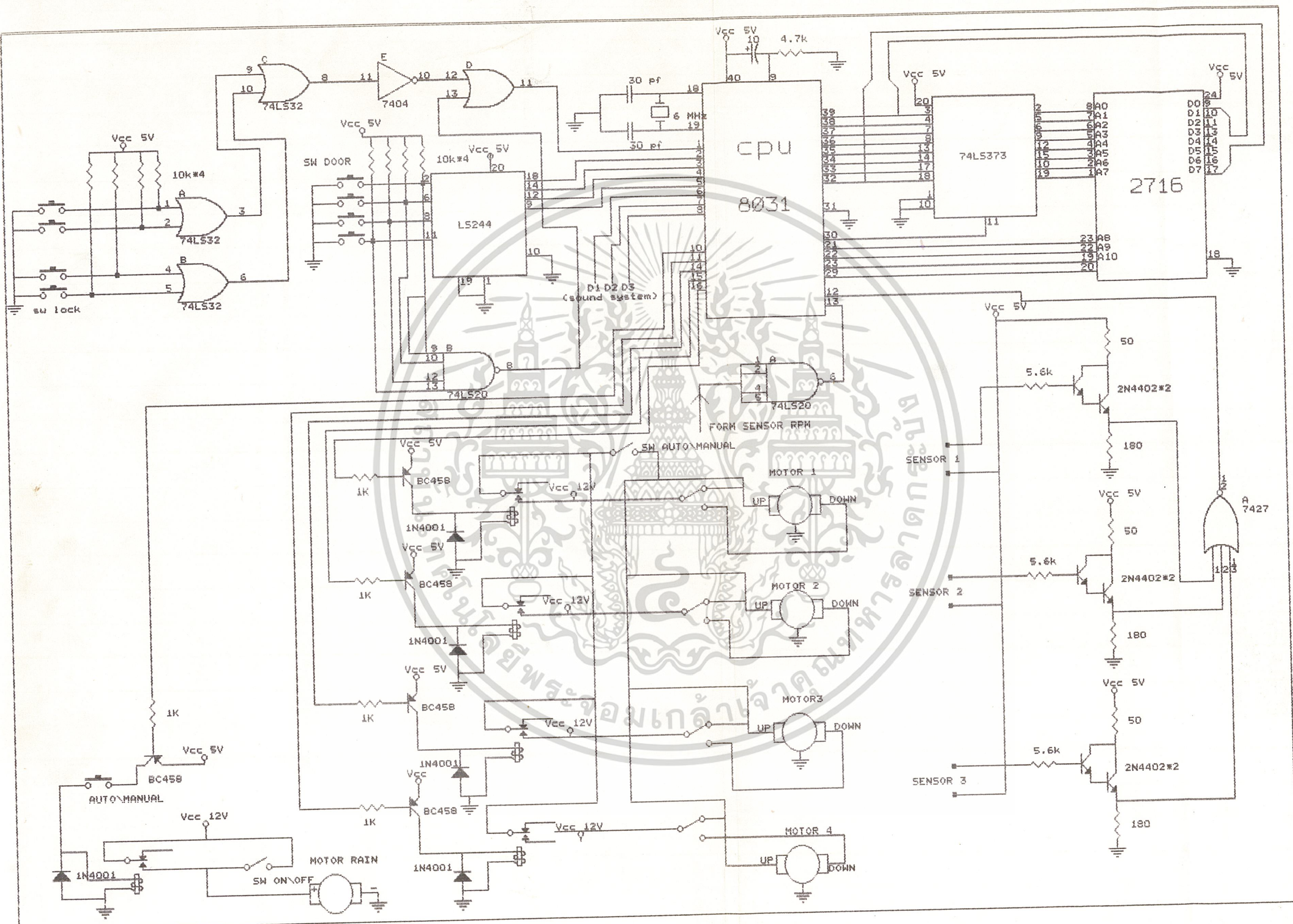
ขาของพอร์ต ๓ ทั้งหมดกับอีก ๒ บิทของพอร์ต ๑ (๘๐๕๒) จะทำงานหลายหน้าที่ ดังรายละเอียดในรูป ๒

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ ๓ โครงสร้างแต่ละบิทของพอร์ต

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

*P1.0	T2	{TIMER/COUNTER 2 EXTERNAL INEUT}
*P1.1	T2EX	{TIMER/COUNTER 2 CAPTURE/RELOAD TRIGGER}
P3.0	RXD	{SERIAL INPUT PORT}
P3.1	TXD	{SERIAL OUTRUT PORT}
P3.2	INTO	{EXTERNAL INTERUPT}
P3.3	TNT1	{EXTERNAL INTERUPT}
P3.4	TO	{TIMER/COUNTER 0 EXTERNAL INPUT}
P3.5	T1	{TIMER/COUNTER 1EXTERNAL INPUT}
P3.6	WR	{EXTERNAL DATA MEMORY WRITE STROBE}
P3.7	RD	{EXTERNAL DATA MEMORY READ STROBE}

หมายเหตุ *P1.0 และ P1.1 สงวนไว้สำหรับ 8052

โครงสร้างของพอร์ต

ในรูป ๗แสดงโครงสร้างของ I/O ของแต่ละพอร์ตโดยแสดงเพียงพอร์ตละบิทส่วนของพอร์ตแลตช์ใช้ D-FLIP FLIP FLOP จะถูกป้อนกลับไปที่บัสภายในเพื่อตอบสนองต่อสัญญาณ'READ LATCH" จาก CPU เมื่อ CPU ต้องการอ่านพอร์ต ส่วนสัญญาณที่ขาภายนอกของพอร์ตจะถูกต่อเข้าขาข้อมูลภายใน และขาพอร์ตภายนอกนี้จะถูกอ่านโดยตอบสนองต่อสัญญาณ "READ PIN" ซึ่งอ่านมาจาก CPU คำสั่งบางคำสั่งจะอ่านพอร์ตโดยอ่านจาก "READ LATCH" (ดูรูปประกอบ) ส่วนคำสั่งอื่น ๆ ใช้สัญญาณ "READ PIN" ดูรายละเอียดคำสั้งที่อ่านข้อมูลจาก READ LATCH ในรูป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

โครงสร้างแต่ละบิทของพอร์ต

ANL	{LOGIC AND, e.g., ANL P1, A}
ORL	{LOGIC OR, e.g., ORL P2, A}
XRL	{LOGIC EX-OR e.g., XRL P3, A}
JBC	{JUMP IF BIT = 1 AND CLEAR BIT, e.g., JBC P1.1, LAEL}
CPL	{COMPLEMENT BIT, e.g., CPL P3.0}
INC	{INCREMENT, e.g., INC P2}
DEC	{DECREMENT, e.g., DEC P2}
DJNZ	{DECREMENT AND JUMP IF NOT ZERO,e.g., DJNZ P3,LABEL}
MOV PX.Y,C	{MOVE CARRY BIT TO BIT Y OF PORT X}
CLR RX.Y	{CLEAR BIT Y OF PORT X}
SET RX.Y	{SET BIT Y OF PORT X}

คำสั่งที่อ่านจาก READ PIN

เหตุที่บางคำสั่งต้องอ่านพอร์ตจาก "READ LATCH" เพราะในบางกรณี CPU อาจเข้าใจสัญญาณที่ขาของพอร์ตผิดพลาด ตัวอย่าง PORT ๑ ขาที่ ๑ ถูกต่ออยู่กับขา BASE ของ TRANSISTOR และให้สัญญาณทางออกเป็น HIGH แก่ขา BASE ของทรานซิสเตอร์ ในขณะที่ทรานซิสเตอร์นำกระแสจะทำให้แรงดันที่ขา BASE เหลือโดยประมาณ ๐.๖ โวลท์ ซึ่งขา BASE ถูกต่ออยู่กับพอร์ตเมื่อ CPU ทำการอ่านพอร์ตจะทำให้เข้าใจสัญญาณที่ขาของพอร์ตผิดไปทั้ง ๆ ที่ขณะนั้นขาของพอร์ตเป็น HIGH อยู่

จากรูป จะเห็นว่าพอร์ต ๐ นอกจากจะเป็น I/O พอร์ตแล้วยังเป็นได้ทั้งแอดเครสบัสและบัสข้อมูล ส่วนพอร์ต ๒ เป็นทั้ง I/O พอร์ตและแอดเครส ซึ่งควบคุมโดยสัญญาณภายในชิพในขณะทำการติดต่อกับหน่วยความจำภายนอก SFR ของพอร์ต ๒ จะไม่เปลี่ยนค่าหลังจากให้แอดเครสแล้ว แต่ SFR และพอร์ต ๐ จะเปลี่ยนเป็น HIGH ทั้ง ๘ บิท เพื่อที่จะรับข้อมูลที่จะส่งมาทางบัสข้อมูลของหน่วยความจำ (เนื่องจาก SFR สามารถเลคช์ข้อมูลได้ ถ้าไม่ทำให้เป็น HIGH อาจเกิดจากผิดผลาดในการอ่านข้อมูลจากหน่วยความจำภายนอก)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

พอร์ต ๑, พอร์ต ๒ และพอร์ต ๓ มีการพูลอัพภายใน ส่วนในพอร์ต ๐ จะเป็น OPEN DRAIN OUTRUT แต่ในกรณีที่ใช้พอร์ต ๑, ๒, ๓, เป็นอินพุทจะต้องทำให้เป็น HIGH จากความต้านทางพูลอัพภายใน

ข้อแตกต่างของพอร์ต ๐ ก็คือ ไม่มีการพูลอัพภายใน FET ตัวที่ทำหน้าที่พูลอัพ (ตัวบน) จะใช้เพียงให้เอ๊าท์พุทเป็น ๑ ในขณะที่ติดต่อกับหน่วยความจำภายนอก ในกรณีอื่น ๆ FET ตัวนี้จะถูกทำให้คัทออฟ ฉะนั้นพอร์ต ๐ ที่ใช้เป็นเอ๊าท์พุทพอร์ตจะเป็น OPENDRAIN

การเขียน ๑ (FFH) ไปที่พอร์ต ๐ จะทำให้ FET ทั้ง ๒ ตัวหยุดทำงานผลก็คือทำให้ขาพอร์ต ๐ ลอยและสามารถใช้เป็นขาอินพุทแบบความต้านหน่วยสูง(HI-Z INPUT) การรีเซ็ท ๘๐๓๑ จะทำให้ทุกพอร์ตมีระดับสัญญาณเป็น HIGH

การติดต่อกับหน่วยความจำภายนอก

การติดต่อกับหน่วยความจำภายนอกกระทำได้ ๒ แบบ คือ ติดต่อกับโปรแกรมภายนอกกับติดต่อกับหน่วยความจำภายนอกที่เก็บข้อมูล การติดต่อกับโปรแกรมภายนอกจะใช้สัญญาณ PSEN {PROGRAM STORE ENABLE} เป็นสัญญาณอ่านโปรแกรมภายนอก ส่วนการติดต่อกับหน่วยความจำภายนอกที่เก็บข้อมูลจะใช้ RD {P3.7}และ WR {P3.6} เหมือนกับ CPU ทั่ว ๆ ไป

การติดต่อกับข้อมูลภายนอกจะใช้ได้ทั้ง ๑๖ บิท {MOVX @DPTR} หรือเป็นแบบ ๘ บิท {MOVX @RI} เมื่อไรก็ตามที่ต้องใส่แอดเดรสขนาด ๑๖ บิท แอดเดรสไบท์สูง (A8-A15) จะออกทางพอร์ต ๒

เมื่อติดต่อกับหน่วยความจำของข้อมูลภายนอกแบบ ๘ บิท {MOVX A, @RI} จะไม่มีผลกระทบกับ SFR ของพอร์ต ๒ คือ SFR ของพอร์ต ๒ จะยังคงแลตช์ข้อมูลเดิมเอาไว้ ประโยชน์คือ สามารถใช้พอร์ต ๒ กำหนดหน้า {PAGE} ของหน่วยความจำ

การที่จะติดต่อกับหน่วยความจำภายนอกได้ขึ้นอยู่กับสภาวะอยู่ ๒ ประการคือ

๑.เมื่อต่อขา EA ของ ๘๐๓๑ ลงกราวด์

๒.เมื่อไรก็ตามที่โปรแกรมเค๊าท์เตอร์ {PC} มีค่ามากกว่า OFFFH

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สัญญาณ PSEN

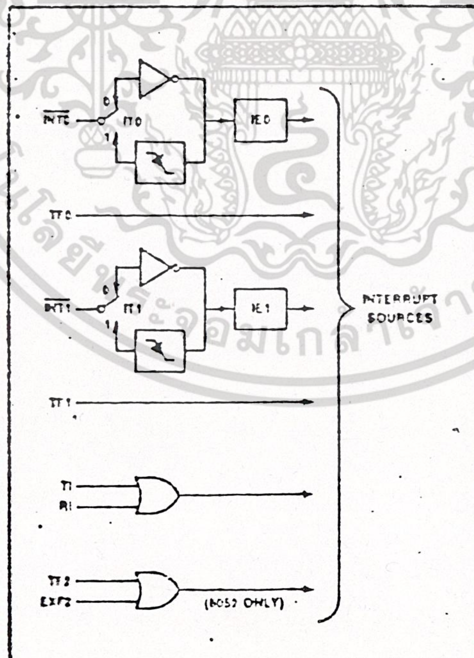
เมื่อมีการเฟ่ซคำสั่งจากโปรแกรมภายนอก สัญญาณ PSEN จะแอกทีฟ ๒ ครั้งทุก ๆ ๑ แมชีนไซเคิล (ยกเว้นคำสั่ง MOVX)

สัญญาณ ALE

จะเป็นขาสโตรปในการแลตช์แอ็คเดรสไบท์ต่ำ เมื่อทำการติดต่อกับหน่วความจำภายนอก ALE จะแอคทีฟ ๒ ครั้ง ทุก ๆ แมชีนไซเคิล ยกเว้นขณะคำสั่งที่ติดต่อกับข้อมูลในหน่วยความจำภายนอก {AOVX} ฉะนั้นในกรณีที่ระบบไม่ได้ใช้หน่วยความจำนอก {MOVX} การแอคทีฟของ ALE จะคงที่ในอัตรา ๑/๖ ขอความถี่ออสซิลเลเตอร์ เมื่อใช้เป็น clock ให้กับวงจรภายนอกได้ การใช้งานบางกรณีต้องการที่จะใช้ทั้งโปรแกรมและข้อมูลในเพจเดียวกัน {64} เราสามารถทำได้โดยการรวมสัญญาณ PSEN และ RD โดยใช้ AND GATE

อินเตอร์รัพท์

๘๐๓๑ จะมีแหล่งกำเนิดสัญญาณอินเตอร์รัพท์ ๕ อย่าง ดังในรูป



รูปที่ ๑.๒ INTERUPT SOURCE

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

มารถจะเกิดขึ้นหรืออินเตอร์รัพท์ที่ค้างอยู่สามารถยกเลิกได้โดยซอร์ฟแวร์

แหล่งกำเนิดสัญญาณอินเตอร์รัพท์แต่ละตัวสามารถจะ ENABLE หรือ DISABLE โดยการเซ็ทหรือเคลียร์บิทที่อยู่ในรีจิสเตอร์ IE ข้อสังเกตุในรีจิสเตอร์ IE บิทที่ ๗ คือ EA จะเป็นตัวควบคุมการ ENABLE หรือ DISABLE ของสัญญาณอินเตอร์รัพท์ทุกสัญญาณ ฉะนั้นเมื่อต้องการใช้อินเตอร์รัพท์ต้องไม่ลืมที่จะเซ็ทบิท EA ด้วย หลักจากนั้นจึงทำการ ENABLE สัญญาณอินเตอร์รัพท์ที่ต้องการ

(MSB) (LSB)

EA	X	ET2	ES	ET1	EX1	ET0	EX0

Symbol	Position	Function
EA	IE.7	disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
—	IE.6	reserved
ET2	IE.5	enables or disables the Timer 2 overflow or capture interrupt. If ET2 = 0, the Timer 2 interrupt is disabled.
ES	IE.4	enables or disables the Serial Port interrupt. If ES = 0, the Serial Port interrupt is disabled.
ET1	IE.3	enables or disables the Timer 1 Overflow interrupt. If ET1 = 0, the Timer 1 interrupt is disabled.
EX1	IE.2	enables or disables External interrupt 1. If EX1 = 0, External interrupt 1 is disabled.
ET0	IE.1	enables or disables the Timer 0 Overflow interrupt. If ET0 = 0, the Timer 0 interrupt is disabled.
EX0	IE.0	enables or disables External interrupt 0. If EX0 = 0, External interrupt 0 is disabled.

รูป IE : INTERURT ENABLE REGISTER

ลำดับความสำคัญของการอินเตอร์รัพท์

แหล่งสัญญาณอินเตอร์รัพท์แต่ละสัญญาณสามารถโปรแกรมได้ว่าให้เป็นลำดับความสำคัญสูงหรือลำดับความสำคัญต่ำจะถูกอินเตอร์รัพท์ด้วยสัญญาณอินเตอร์รัพท์ที่มีความสำคัญสูงกว่าและสำคัญความสำคัญสูงจะไม่ถูกอินเตอร์รัพท์โดยสัญญาณอินเตอร์รัพท์ที่มีความสำคัญต่ำกว่า

ถ้ามีการอินเตอร์รัพท์ด้วยลำดับความสำคัญเท่ากันมากกว่า ๑ สัญญาณ CPU จะทำการตรวจ{POLLING} และตัดสินใจว่าจะให้บริการกับสัญญาณอินเตอร์รัพท์ตัวใด ฉะนั้นในแต่ละลำดับความสำคัญยังมีการจัดลำดับความสำคัญไว้อีกดังรายละเอียดข้างล่าง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

	SOURCE	RIORITY WITHIN LEUEL
1.	IEO	{HIGHEST}
2.	TFO	
3.	TE1	
4.	TF1	
5.	RI + TI	
6.	TF2 + EXF2	{LOWEST}

หมายเหตุ ลำดับความสำคัญนี้ใช้เฉพาะเมื่อมีสัญญาณอินเตอร์รัพท์ในลำดับความสำคัญเท่ากันมากกว่า ๑ สัญญาณ

การทำงานของสัญญาณอินเตอร์รัพท์

แฟลกของสัญญาณอินเตอร์รัพท์จะถูกสุ่มตัวอย่างในเสตท ๕ เฟสที่ ๒ ของทุก ๆ แมชีนไซเคิลและจำทำการตรวจ {POLLING} การอินเตอร์รัพท์จาก ๕ แหล่งสัญญาณในแมชีนไซเคิลต่อมาถ้าผลของการสุ่มตัวอย่างพบว่า แฟลกอินเตอร์รัพท์ถูกเซ็ทในเสตทที่ ๕ เฟสที่ ๒ ของแมชีนไซเคิลที่ผ่านมาแล้วจะมีการเรียกไปยังส่วนของโปรแกรมบริการอินเตอร์รัพท์ หากว่าไม่ถูกขัดขวางด้วยสภาวะใดสภาวะหนึ่งดังต่อไปนี้

๑.กำลังทำคำสั่งอยู่ในโปรแกรมบริการอินเตอร์รัพท์ที่มีความสำคัญเท่ากันหรือสูงกว่า

๒.ไม่ใช่ไซเคิลสุดท้ายของคำสั่งที่กำลังปฏิบัติอยู่

๓.คำสั่งที่ปฏิบัติอยู่ที่นั้นคือ RETI หรือ คำสั่งที่ติดต่อกับรีจิสเตอร์ IE หรือ IP

ในสภาวะตามข้อ ๒ เพื่อเป็นการประกันว่าคำสั่งที่ปฏิบัติถึงไซเคิลสุดท้ายแล้วจะไม่ถูกอินเตอร์รัพท์จนกว่าจะปฏิบัติคำสั่งนั้นจนจบเสียก่อน

ตามข้อ ๓ นั้นในภรณีที่ CPU กำลังทำคำสั่ง RETI หรือกำลังติดต่อกับ IE หรือ IP ตัวในตัวหนึ่งอยู่แล้วเกิดอินเตอร์รัพท์ขึ้น CPU จะยอมให้มีการอินเตอร์รัพท์แต่ต้องปฏิบัติอย่างน้อยอีก ๑ คำสั่งหลังจากทำคำสั้ง IF, IP หรือ RETI ตัวอย่างเช่น ถูกอินเตอร์รัพท์ในขณะที่กำลังทำคำสั่ง RETI หน่วยประมวลผลจะสั่งแอดเดรสคืนให้ PC หลังจากคำสั่ง RETI และปฏิบัติอีก ๑ คำสั่งในโปรแกรมหลักต่อจากนั้นจึงจะตอบสนองการอินเตอร์รัพท์ CPU รับรู้การอินเตอร์รัพท์โดยกระโดดไปทำโปรแกรมหลักต่อจากนั้นจึงจะตอบสนองการอิน

ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เตอร์รัพท์ CPU รับรู้การอินเตอร์รัพท์โดยกระโดดไปทำโปรแกรมบริการอินเตอร์รัพท์ ในบางกรณีจะไม่เคลียร์แฟลกที่กำเนิดสัญญาณอินเตอร์รัพท์นั้น กรณีที่ไม่เคลียร์แฟลกส่วนของโปรแกรมของผู้ใช้จะต้องมีคำสั่งเคลียร์แฟลกเอง เช่น การอินเตอร์รัพท์ที่เกิดจากพอร์ตอนุกรม ส่วนสัญญาณอินเตอร์รัพท์จากภายนอก แฟลกจะถูกเคลียร์ให้ถ้าเป็นการโปรแกรมให้รับการอินเตอร์รัพท์แบบการเปลี่ยนแปลงของของสัญญาณ การตอบสนองสัญญาณอินเตอร์รัพท์ CPU จะกระโดดไปที่ตำแหน่งของโปรแกรมบริการอินเตอร์รัพท์ตามชนิดของอินเตอร์รัพท์ ดังนี้

SOURCE	VECTOR ADDRESS
IEO	0003H
TFO	000BH
IE1	0013H
TF1	001BH
R1+T1	0023H
TF2+EXF2	002BH {80322/52}

การอินเตอร์รัพท์จากภายนอก

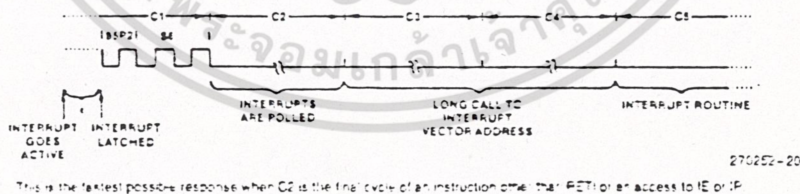
การอินเตอร์รัพท์ภายนอกกระทำได้ ๒ อินพุทคือ INTO และ INT1 โดยสามารถโปรแกรมให้เป็นแบบการเปลี่ยนแปลงขอบสัญญาณหรือเป็นแบบระดับสัญญาณก็ได้ โดยเซ็ทหรือเคลียร์บิท IT1 หรือ ITO ในรีจิสเตอร์ TCON ถ้า ITX =0 เป็นการอินเตอร์รัพท์แบบระดับสัญญาณ {LOW} ถ้า ITX =1 จะเป็นการรับอินเตอร์รัพท์แบบการเปลี่ยนแปลงขอบของสัญญาณและในกรณีนี้ที่ขา INTX จะต้องได้รับสัญญาณ HIGH 1 ไซเคิล และ LOW ในไซเคิลต่อไปส่วนของอินเตอร์รัพท์จะเซ็ท IEx ใน TCON ฉะนั้นสัญญาณที่เป็น HICH และ LOW ที่กล่าวมาจะต้องมีค่าอย่างน้อย ๑๒ คาบเวลาของความถี่ออสซิลเลเตอร์ และ IEx จะถูกเคลียร์เมื่อโปรแกรมกระโดดไปทำงานในส่วนของเซอร์วิสรูทีนโดยอัตโนมัติ ถ้าสัญญาณอินเตอร์รัพท์จากภายนอกเป็นการอินเตอร์รัพท์แบบระดับสัญญาณ {LOW} วงจรอินเตอร์รัพท์ภายนอกต้องรักษาระดับสัญญาณ ๐ จนกว่าส่วนบริการอินเตอร์รัพท์จะทำงาน และต้องถอนตัวจากการอินเตอร์รัพท์ก่อนที่ CPU จะเสร็จสิ้นโปรแกรมบริการอินเตอร์รัพท์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เวลาในการตอบสนองการอินเตอร์รัพท์

ถ้าการอินเตอร์รัพท์จากภายนอกเกิดขึ้นในภาวะปกติ CPU จะใช้เวลาตั้งแต่เซ็ทอินเตอร์รัพท์แฟลก ตรวจสอบ{POLLING} ไปจนถึงกระโดดไปทำคำสั่งของซับรูทีน อย่างน้อยที่สุด ๓ แมชีนไซเคิล ในบางสภาวะจะใช้เวลามากว่า ๓ แมชีนไซเคิลถ้าผลของอินเตอร์รัพถูกขัดขวางด้วยสภาวะใดสภาวะหนึ่งใน ๓ ข้อจากที่กล่าวมาแล้วข้างต้น เช่น ถ้าผลของการอินเตอร์รัพท์ในขณะที่ CPU ทำคำสั้งอยู่ซับรูทีนอินเตอร์รัพท์ที่ลำดับความสำคัญเท่ากันหรือสูงกว่าเวลาจะมากหรือน้อยขึ้นอยู่กับโปรแกรมในซับรูทีของอินเตอร์รัพท์ที่กำลังทำอยู่ ถ้าคำสั่งที่กำลังดำเนินนั้นไม่ใช่แมชีนไซเคิลสุดท้ายเวลาในการตอบสนองการอินเตอร์รัพท์จะมากขึ้นแต่จะไม่เกิน ๓ แมชีนไซเคิลเพราะว่าคำสั่งที่ยาวที่สุด (คูณและหาร) จะยาวเพียง ๔ แมชีนไซเคิล และถ้าคำสั่งที่กำลังดำเนินอยู่เป็นคำสั่ง RETI หรือคำสั้งติดต่อกับรีจิสเตอร์ IE หรือ IP เวลาในการตอบสนองการอินเตอร์รัพท์จะเพิ่มขึ้นแต่ไม่มากว่า ๕ แมชีนไซเคิล (จะต้องทำอย่างที่สุดอีก ๑ แมชีนไซเคิลสำหรับทำให้คำสั่ง {PETI} จบสมบูรณ์และกับอีก ๔ แมชีนไซเคิลสำหรับคำสั่งที่ยาวที่สุดอีก ๑ คำสั่ง)

สรุป สัญญาณอินเตอร์รัพท์เดี่ยว (โดยไม่ซ้อนกับอินเตอร์รัพท์อื่น) จะใช้เวลาในการตอบสนองมากกว่า ๓ แมชีนไซเคิล และน้อยกว่า ๘ แมชีนไซเคิล



รูป ๕ INTERUPT RESPONSE TIMONG DIAGRAM

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การรีเซ็ท

สัญญาณรีเซ็ทเป็นสัญญาณอินพุททางขา ๙ การรีเซ็ทจะสมบูรณ์ต้องรักษาระดับ HIGH อย่างน้อยที่สุด ๒ แมชีนไซเคิล (๒๔ คาบเวลาของออสซิลเลเตอร์) การรีเซ็ทภายในตัว CPU จะเริ่มในระหว่างไซเคิลที่ ๒ นับตั้งแต่ขา RST เป็น HIGH ผลของการรีเซ็ทจะ

มีผลกับรีจิสเตอร์ดังต่อไปนี้

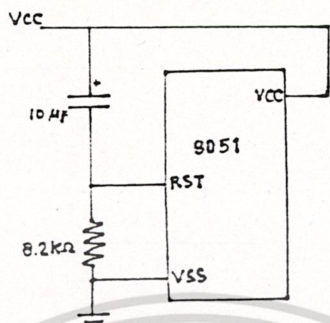
REGISTOR	CONTENT
PC	0000H
ACC	00H
B	00H
PSW	00H
SP	07H
DPTR	0000H
PO-P3	FFH
IP	{XXOOOOOO}
IE	{OXOOOOOO}
TMOD	00H
TCON	00H
THO	00H
TLO	00H
TH1	00H
SCON	00H
SBUF	00H
PCON	00H

RAM ภายในจะไม่ถูกเคลียร์เมื่อ CPU ถูกรีเซ็ท

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การรีเซ็ทเมื่อเปิดเครื่อง

เมื่อจ่ายไฟเข้าระบบควรจะมีการรีเซ็ท CPU ก่อนเพื่อรอให้ทั้งระบบอยู่ในสภาวะพร้อมที่จะทำงานซึ่งทำได้โดยต่อ C ขนาด 10uF จาก VCC มาที่ขา ๙ {RST} และจากขา ๙ ต่อ R ขนาด 8.3K ลงกราวด์



Power on Reset Clrcult

IDLE MODE

๘๐๓๑ มีคำสั่งเซ็ท PCON.O ซึ่ง CPU ทำคำสั่งนี้จะเป็นคำสุดท้ายก่อนจะเข้าสุดท้ายก่อนจะเข้าสู่ IDLE โหมด (ไม่ทำงาน) IDLE โหมดนี้สัญญาณนาฬิกาภายในจะหยุดส่งให้ CPU แต่ยังจ่ายสัญญาณนาฬิกาให้กับอินเตอร์รัพท์, TIMER และพอร์ตอนุกรมสาถานะต่าง ๆ ของ CPU ถูกเก็บไว้ และรีจิสเตอร์ต่าง ๆ ยังคงรักษาข้อมูลเดิมไว้ในระหว่างอยู่ใน IDLE โหมด การที่จะทำให้ COP กลับสู่โหมดปกติทำได้ ๒ อย่างคือ ทำการอินเตอร์รัพท์จากภายนอกเมื่อ CPU ถูกอินเตอร์รัพท์จะให้บริการอินเตอร์รัพท์เมื่อจบบริการอินเตอร์รัพท์ CPU จะกลับมาทำคำสั่งต่อไปของโปรแกรมซึ่งต่อจากคำสั่งที่ทำ IDLE โหมด การประยุต์ให้งานเราอาจใช้ประโยชน์จาก GF!, GFO ซึ่งเป็นแฟลกที่ใช้ในจุดประสงค์ทั่ว ๆ ไปโดยการเซ็ทบิทใดบิทหนึ่งหรือทั้งสองก่อนคำสั่ง IDLE โหมดและเมื่อถูกอินเตอร์รัพท์ในโปรแกรมบริการอินเตอร์รัพท์ตรวจดูว่าเป็นการอินเตอร์รัพท์หลังจากทำคำสั่ง IDLE โหมดหรือไม่เพื่อที่จะให้บริการที่ถูกต้อง อีกวิธีหนึ่งที่จะออกจาก IDLE โหมดได้คือทำการรีเซ็ท CPU ทางขารีเซ็ท

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รายละเอียดของขาต่าง ๆ

VCC : ต่อกับไปเลี้ยงของระบบ (5VDC)

VSS : กราวด์

PORTO : พอร์ด ๐ เป็นอินพุท/เอ๊าท์พุทพอร์ตแบบ ๘ บิท{OPEN DRAIN} สามารถรับกระแส SINK จาก LS TTL ได้ ๘ ตัว ถ้าเราเขียน ๑ ไปที่พอร์ต ๐ จะทำให้พอร์ต ๐ เป็น HI-Z อินพุท พอร์ต ๐ สามารถ MULTIPLEX ระหว่างบัสข้อมูลกับแอดเดรสไบท์ต่ำเมื่อใช้คิดต่อกับหน่วยความจำภายนอก

PORT1 : พอร์ต ๑ เป็น ๘ บิท สองทิศทางมีการพูลอัพภายใน, พอร์ต ๑ นี้ขับ TTL {LS} ได้ ๔ ตัว เมื่อเขียนค่า 1H ไปที่พอร์ต ๑ ขาของพอร์ต ๑ จะเป็น HIGH โดยมีการพูลอัพภายในและสามารถใช้เป็นอินพุทพอร์ตได้

PORT2 : เป็นอินพุท/เอ๊าท์พุทพอร์ตแบบ ๒ ทิศทางมีการพูลอัพภายในสามารถขับ TTL {LS} ได้ ๔ ตัว และพอร์ต ๒ นี้จะให้แอดเครส

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ไบท์สูงในขณะติดต่อกับหน่วยความจำภายนอก

PORT3 : พอร์ต ๓ เป็นอินพุท/เอ๊าท์พุทพอร์ตแบบ ๒ ทิศทางมีการพูดอัพภายในพอร์ต ๓ ขับ TTL {LS} ได้ ๔ ตัว และยังสามารถใช้ทำงานในลักษณะพิเศษดังรายละเอียดข้างล่าง

PORT PIN	ALTERNATE FUNCTION
P3.0	RXD {SERIAL INPUT PORT}
P3.1	TXD {SERIAL OUTPUT PORT}
P3.2	INTO {EXTERNAL INTERUPTO}
P3.3	INT1 {EXTERNAL INTERUPT1}
P3.4	TO {TIMER O EXTERNAL TNPUT}
P3.5	WR {EXTERNAL DATA MEMORY WRITE STROBE}
P3.7	RD {EXTERNAL DBTA MEMORY READ STROBE}

RST : ขารีเซ็ทสัญญาณ HIGH ที่ขารีเซ็ทนี้นาน ๒ แมชีนไซเคิลจะเป็นการรีเซ็ท CPU

ALE/PROG : ADDRESS LATCH ENABLE พัลส์สำหรับแลตช์แอดเดรสไบท์ต่ำในระหว่างการติดต่อกับหน่วยความจำภายนอก ALE นี้ยังจ่ายความถี่คงที่ ๑/๖ ของความถี่ออสซิลเลเตอร์ (ถ้าไม่ติดต่อกับหน่วยความจำภายนอก) และ ALE อีกหน้าที่หนึ่งคือเป็นอินพุทรับพัลส์ {LOW} ในระหว่างโปรแกรม EPROM (๘๗๕๑)

PSEN : เป็นขาสัญญาณ READ STROBE ในขณะอ่านโปรแกรมจากภายนอก ESEN จะแอคทีฟ ๒ ครั้งต่อ ๑ แมชีนไซเคิล

EA/UPP : เป็นขาอินพุทของอินเวอร์เตอร์ของภาคขยายความถี่

XTAL 2 : เป็นขาเอ๊าท์พุทของอินเวอร์เตอร์ของภาคขยายความถี่

การจัดหน่วยความจำ

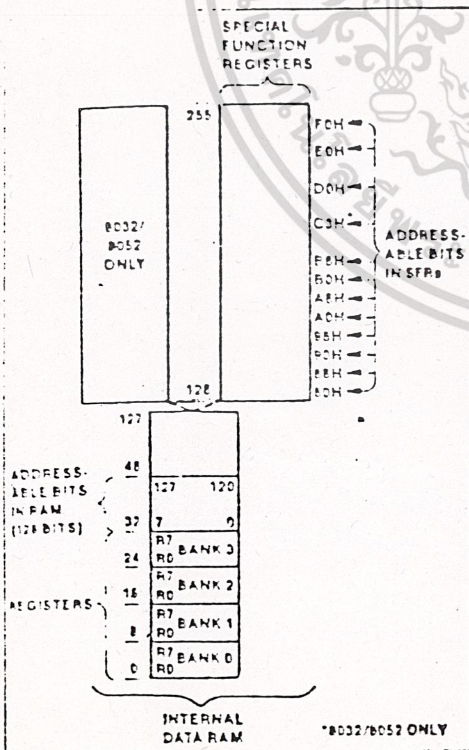
โปรกรมแมมโมรี : ไมโครคอนโทรเลอร์ตระกูล MCS-51 จะแบ่งหน่วยความจำเป็นส่วนของโปรแกรมแมมโมรีและส่วนของข้อมูล {DATA} อย่างละ 64K ถ้า EA ต่อ HIGH

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

จะเป็นการรันโปรแกรมที่อยู่ภายในตัวโดยแอดเครสจะไม่เกิน OFFFH และถ้า EA ต่อLOW จะเป็นการรันโปรแกรมภายนอก ที่แอดเคตส ๐๐ ถึง 23H จะเป็นส่วนของการอินเตอร์รัพท์

หน่วยความจำเก็บข้อมูล

แอดเครสของข้อมูลจะประกอบด้วยหน่วยความจำภายในและภายนอกโดยที่หน่วยความจำสำหรับข้อมูลภายนอกติดต่อได้โดยคิดสั่ง MOVE และมี DETR เป็นตัวชี้ หน่วยความจำภายในถูกแบ่งเป็น๓ ส่วน คือ แอดเครสด้านต่ำ ๑๒๘ ไบท์ {RAM} และSER {SRECIAL EUNCTION REGISTOR} ซึ่งแท้ที่จริงก็คือ RAM อีก ๑๒๘ ไบท์นั้นเอง แต่การเข้าถึงหน่วยความจำใน SFR นี้แตกต่างจาก ๑๒๘ ไบล์ล่าง หน่วยความจำทางด้านต่ำ {LOWER RAM 0-31H} ยังแบ่งเป็น 4 BANK โดยที่สามารถเข้าถึงได้ครั้งละ 1BANK เท่านั้นแต่ละ BANK เลือกโดยเซ็ทบิท RO,R1 ใน RSW หน่วยความจำอีก ๑๖ ไบล์ ตำแหน่ง 20H ถึง 47H ประกอบด้วยหน่วยความจำที่สามารถเข้าถึงแบบบิทได้อีก ๑๒๘ บิท



RAM Byte	(MSB)							(LSB)	
7FH									127
30H									
2FH	7F	7E	7D	7C	7B	7A	79	78	47
2EH	77	76	75	74	73	72	71	70	46
2DH	6F	6E	6D	6C	6B	6A	69	68	45
2CH	67	66	65	64	63	62	61	60	44
2BH	5F	5E	5D	5C	5B	5A	59	58	43
2AH	57	56	55	54	53	52	51	50	42
29H	4F	4E	4D	4C	4B	4A	49	48	41
28H	47	46	45	44	43	42	41	40	40
27H	3F	3E	3D	3C	3B	3A	39	38	39
26H	37	36	35	34	33	32	31	30	38
25H	2F	2E	2D	2C	2B	2A	29	28	37
24H	27	26	25	24	23	22	21	20	36
23H	1F	1E	1D	1C	1B	1A	19	18	35
22H	17	16	15	14	13	12	11	10	34
21H	0F	0E	0D	0C	0B	0A	09	08	33
20H	07	06	05	04	03	02	01	00	32
1FH–18H	Bank 3								31–24
17H–10H	Bank 2								23–16
0FH–08H	Bank 1								15–8
07H–00H	Bank 0								7–0

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

Intemal Bata Memory Address Space   Speclal Functlon Bit Addressable Lecltions

ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ADDRESSING MODES

๘๐๓๑ แบ่งการเข้าถึงหน่วยความจำได้ ๕ แบบ ดังรูป

---

1. REGISTOR ADDRESSING
   - RO-R7
   - ACC, B, CY {BIT}, DPTR
2. DIRECT ADDRESSING
   - LOWER 128 BYTES OF INTERNAL RAM
   - SREICIAL FUNCTION REGISTER
3. REGISTOR INDIRECT ADDRESSING
   - INTERNAL RAM @RI,
   - EXTERNL DATA MEMORY {@RI, RO, @DPIR}
4. IMMEDIATE ADDRESSING
   - PROGRAM MEMORY
5. BASE-REGISTOR PLUS INDEX-REGISTOR INDIRECT ADDRESS
   - PROGRAM MEMORY {@DPTR+A, @PC+A}

รูป ๖ ADDRESSING MODE

REGISTOR ADDRESSING

เป็นการติดต่อกับรีจิสตอร์ทั้ง ๘ ตัวในแต่ละ BANK โดยใช้ ๓ บิทล่างของ OP-CODE เป็นตัวกำหนดรีจิสเตอร์ที่จะทำการติดต่อด้วย ACC, B, DRTR และ CY และการประมวลผลทางลูลีนถือว่าอยู่ในโหมด REGISTOR ADDRESSING ตัวอย่างเช่น MOV A, RO

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

DIRECT ADDRESS

เป็นเพียงวิธีเดียวที่จะเข้าถึง SPE {SPECIAL RUNCTION REGISTOR} ได้ การติดต่อกับ RAM 128 ไบล์ล่างก็ใช้โหมดนี้ด้วย ตัวอย่างเช่น MOV A, SBUF

REGISTOR-INDIRECT ADDRESSING

ในโหมดนี้ใช้ค่าที่อยู่ในรีจิสเตอร์ RO หรือ R1 (ใน BANK ที่เลือกไว) เป็นตัวชี้ไปยังตำแหน่งต่าง ๆ ภายใน ๑๕๖ ไบท์ (RAM 128 ไบล์ล่างหรือ ๒๕๖ ไบล์ล่างของหน่วยความจำภายนอก

ข้อสังเกต คือ SFR ไม่สามารถติดต่อได้ด้วยวิธี REGISTOR-INDIRECT นี้ ส่วนการติดต่อกับหน่วยความ'จำข้อมูลภายนอก 64K นั้นใช้ตัวนี้ขนาด ๑๖ บิท {DPIR} คำสั้ง PUSH และ POP ก็ทำงานในโหมดนี้ด้วย (ใช้ SP เป็นตัวชี้) ตัวอย่าง MOV A, @RO

IMMEDIATE ADDRESSING

โหมดนี้อนุญาติให้ใช้ค่าคงที่เป็นส่วนหนึ่งของ OP-CODE

ตัวอย่าง MOV A, #01H

MOV DPIR, #1234H

BASE-REGISTOR PLUS INDEX REGISTOR-INDIRECT ADDRESSING

ในโหมดนี้จะใช้ค่าในแอคคิวมูเลเตอร์ A บวกกับเบสรีจิสเตอร์ เช่น DPIR หรือ PC ประโยชน์ของโหมดนี้คือใช้ในการหาค่าที่ต้องการจากตาราง {TABLE}

ตัวอย่าง MOVC A, @A+PC

MOVC A, @A+DPTR

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูป ๓ SPECIAL RENCTION REGISTOR BIT ADDRESS

Direct Byte Address (MSB)	Bit Addresses							(LSB)	Hardware Register Symbol
0FFH									
0F0H	F7	F6	F5	F4	F3	F2	F1	F0	B
0E0H	E7	E6	E5	E4	E3	E2	E1	E0	ACC
0D0H	D7	D6	D5	D4	D3	D2	D1	D0	PSW
0B8H	—	—	—	BC	BB	BA	B9	B8	IP
0B0H	B7	B6	B5	B4	B3	B2	B1	B0	P3
0A8H	AF	—	—	AC	AB	AA	A9	A8	IE
0A0H	A7	A6	A5	A4	A3	A2	A1	A0	P2
98H	9F	9E	9D	9C	9B	9A	99	98	SCON
90H	97	96	95	94	93	92	91	90	P1
88H	8F	8E	8D	8C	8B	8A	89	88	TCON
80H	87	86	85	84	83	82	81	80	P0

270656-3

b.) Special Function Register Bit Addresses

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ชุดคำสั่งของ ๘๐๓๑

๘๐๓๑ มีคำสั้งทั้งหมด ๑๑๑ ประกอบด้วยคำสั้งไบท์เดียว ๔๙ คำสั้ง คำสั้ง ๒ ไบท์ ๔๕ คำสั้ง คำสั้ง ๓ ไบท์ ๑๗ คำสั่ง

เราแบ่งคำสั่งตามหน้าที่การทำงานได้เป็น ๔ แบบ

1. DATA TRANSFER
2. ARITHMETIC
3. LOGIC
4. CONTROL

1.1 DATA TRANSFER ยังแบ่งเป็น

- GENERAL PUROPSE
- ACCUMULATOR-SRECIFIC
- ADDRESS-OBJECT

ทั้งหมดนี้ไม่มีคำสั่งใดที่มีผลกระทบต่อ PSW ยกเว้นการ POP หรือ MOV โดยตรงไปที่ PSW

GENERAL PURPOSE TRANSFER

- MOV จะกระทำเป็นบิทหรือไบท์ก็ได้โดยข้อมูลจะย้ายจากต้นทาง {SOURCE} มายังปลายทาง {DESTINATION}

- PUSH คำสั่งนี้จะเพิ่มค่า SP ขึ้นไปอีก ๑ ก่อนที่จะนำข้อมูลจากต้นทางไปเก็บไว้ในตำแหน่งที่ชี้โดย SP

- POP คำสั่งย้ายข้อมูลจากตำแหน่งที่ชี้โดย SP มายังปลายทางและ SP จะลดค่าลง ๑

ACCUMULATOR SRECIFIC TRANSFER

- XCH {EXCHANGE} แลกเปลี่ยนข้อมูลของต้นทางกับแอคคิวมูเลเตอร์

- XCHD แลกเปลี่ยนค่า ๔ บิทล่างของไบล์ข้อมูลกับข้อมูล ๔ บิทล่างของแอคคิวมูเลเตอร์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- MOVX ทำการย้ายข้อมูลระหว่างแอดคิวมูเลเตอร์กับหน่วยความจำข้อมูลภายในออกโดยที่ตำแหน่งของหน่วยความจำภายนอกกำหนดได้โดยค่าของ DPTR (๑๖ บิท) หรือรีจิสเตอร์ RO, R1 (๘ บิท)

- MOVC ย้ายข้อมูล ๑ ไบท์จากหน่วยความจำของโปรแกรม (ในกรณีแยกหน่วยความจำของข้อมูลต่างหาก) มาไว้ที่ A โดยข้อมูลใน A ก่อนทำคำสั่งนี้จะรวมกัน PC หรือ DPTR เป็นตัวชี้ที่อยู่ของข้อมูลที่ต้องการ

ตัวอย่าง
```
REL-PC :    INC  A
            MOVC A, @A+PC
            RET
            DB   66H
            DB   77H
            DB   88H
            DB   89H
```

ถ้า SUBROUTINE นี้ถูกเรียกโดยที่ A มีค่า = 01H เมื่อออกจากรูทีนจะได้ค่า 77H อยู่ใน A

ARITHMETIC

๘๐๓๑ มีคำสั่งสนับสนุนการคำนวณเบื้องต้น เช่น บวก บล คูณ หาร แบบ ๘ บิท โดยไม่คิดเครื่องหมาย

การบวก {ADDING}

- INC {INCREMENT} เพิ่มค่าในโอเปอร์แรนด์ขึ้นอีก ๑

- ADD บวกค่าของโอเปอร์แรนด์ต้นทางกับแอดคิวมูเลเตอร์และเก็บค่าไว้ในแอดคิวมูเลเตอร์

- DB {DEOIMAL-ADD-ADJUST FOR BCD ADDITION} เป็นการปรับแต่งผลรวมของการบวกเลข BCD และคือผลลัพธ์นั้นให้รีจิสเตอร์ A แฟลก CY จะถูกเซ็ทถ้าผลลัพธ์ที่ได้จากคำสั่ง DA มากกว่า ๙๙ การทำงานคร่าว ๆ ของคำสั่งถ้าค่าที่จะทำการปรับแต่ง ๔ บิทล่างของ A มากกว่า ๙ หรือแฟลก AC = 1 จะทำการบวกด้วย 06H เช่นเดียวกันถ้า ๔ บิทบนมากกว่า ๙ ก็จะบวกด้วย 06H

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การลบ {SUBTRACTION}

- SUBB {SUBTRACT WITH BORROW} ลบค่าในโอเปอร์แรนด์ตัวแรกด้วยโอเปอร์แรนด์ตัวที่ ๒ และลบด้วย ๑ ถ้า CY ถูก SET และคือผลลัพธ์ให้แอดคิวมูเลเตอร์ A

- DEC {DECREMENT} ลดค่าในแอกคิวมูเลเตอร์ลง ๑

การคูณ {MULTIPLICATION}

- MUL เป็นการคูณกันระหว่างรีจิสเตอร์ A กับ B โดยที่ A จะรับค่าไบท์ต่ำและ B รับค่าไบท์สูงของผลการคูณ OV จะถูกเคลียร์ถ้าข้อมูลครึ่งบนสุดของผลลัพธ์เป็น ๐ ส่วน CY จะเคลียร์เสมอ

การหาร {DIVISION}

- DIV เป็นการหารระหว่างรีจิสเตอร์ A โดยรีจิสเตอร์ B และให้ผลลัพธ์เลขจำนวนเต็มในรีจิสเตอร์ A ส่วนเศษของการหารจะอยู่ในรีจิสเตอร์ B การหารด้วยศูนย์จะไม่ให้ค่าผลลัพธ์ใน A และ B และ OV แฟลกจะถูกเซ็ท การทำคำสั่งหารนี้จะมีผลต่อแผลกต่าง ๆ ดังนี้

- CY จะถูกเซ็ทถ้าผลของการหารทำให้เกิดตัวทดขึ้นจากบิทสูงสุด

- AC จะถูกเซ็ทถ้าผลของการหารทำให้เกิดตัวทดจาก ๔ บิทล่างหรือเกิดการยืมบิทบนโดยบิทล่าง

- OV จะถูกเซ็ทจำนวนใด ๆ ถูกหารด้วย ๐ ในกรณีอื่น OV จะถูกเคลียร์ OV ถูกใช้ในการคำนวณแบบ TWO'S COMPLEMENT เพราะว่า OV จะถูกเซ็ทเมื่อมีการใช้จำนวนที่มีเครื่องหมายและไม่สามารถแสดงผลใน ๘ บิทได้

- P {PARITY} ถ้าผลลัพธ์ทำให้เกิดพาริตี้คี่ P จะถูกเซ็ท

LOGIC

๘๐๓๑ สามารถปฏิบัติการทาง LOGIC ได้ทั้งแบบบิทและไบท์

SINGLE-OPERAND OPERATION

- CLR เคลียร์รีจิสเตอร์ A ให้เป็น ๐ หรือเคลียร์บิทใดบิทหนึ่งในรีจิส

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เตอร์หรือหน่วยความจำที่สามารถเข้าถึงแบบบิทให้เป็น ๐ ได้

- SETB เซ็ทบิทต่าง ๆ ถ้าสามารถเข้าถึงแบบบิทได้ให้เป็น ๑

- CPL คอมพลีเมนต์ค่าให้รีจิสเตอร์ A หรือคอมพลีเมนท์โดยตรงกับบิทใดบิทหนึ่งที่สามารถติดต่อแบบบิทได้

- RL, RLC, RR, RRC, SWAP เป็นกลุ่มคำสั่งหมุน {ROTATE} ค่าในรีจิสเตอร์ A SWAP ใช้เปลี่ยนค่ากันระหว่าง ๔ บิทล่างกับ ๔ บิทบนของรีจิสเตอร์ A

TWO OPERAND OPERATIONS

- ANL คำสั่ง LOGIC "AND" แบบบิทหรือไบท์ด้วยรีจิสเตอร์ ๒ ตัว โดยผลลัพธ์เก็บไว้ในรีจิสเตอร์ตัวแรก

- ORL คำสั่ง LOGIC "OR" แบบบิทหรือไบท์โดยรีจิสเตอร์ ๒ ตัว

- XOR คำสั่ง LOGIC "XOR"

CONTROL TRANSFER

๘๐๓๑ แบ่งการควบคุมการส่งผ่านข้อมูลออกเป็น ๓ อย่างคือ CALL แบบไม่มีเงื่อนไข RETURN และ JUMP

UNCONDITIONAL CALLS, RETURN AND JUMP

- ACALL และ LCALL เก็บค่า PC ของคำสั่งต่อไปลงในแสตคและเปลี่ยนการควบคุมระบบให้กับแอดเดรสปลายทางของคำสั่ง CALL

- LCALL เป็นคำสั่ง ๓ ไบท์ ที่สามารถกระโดดไปยังตำแหน่งใด ๆ ภายใน 64K ได้

ACALL เป็นคำสั่ง ๒ ไบท์ ใช้เมื่อการ CALL ไปยังตำแหน่งที่ไม่ไกลเกิน 2K โดยการให้ค่าของ A0 A10 ของตำแหน่งที่จะกระโดดไปมาเข้าโค๊ดในคำสั่ง ACALL

A10 A9 A8 1 0 0 0 1	A7 A6 A5 A4 A3 A2 A1 A0

ข้อความระวัง ถ้าคำสั่ง AJMP อยู่ใน ๒ ไบท์สุดท้ายของ PAGE {2K} เมื่อทำคำสั่งนี้ PAGE จะเปลี่ยนไปเพราะว่า PC จะต้องเพิ่มขึ้นอีก ๒ ไบท์ {AJMP=2 ไบท์ ก่อนที่จะทำคำ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สั่งทำให้การกระโดดไม่ตรงตำแหน่งที่ต้องการเพราะในคำสั่ง AJMP จะเกี่ยวพันกัน A11-A15 ด้วยในกรณีนี้ A11 จะเปลี่ยนไป

- RET กลับสู่โปรแกรมหลักหลังจากการ CALL โดยการ POP ไบท์สูงให้ PC แล้วลดค่า SP ลง ๑ POP ไบท์ต่ำให้ PC และลดค่า SP ลงอีก ๑

- AJMP LJMP และ SJMP คำสั่ง AJMP และ LJMP คล้ายกัน ACALL และ LCALL ส่วน SJMP {SHORT JUMP จะกระโดดได้เพียง ๒๕๖ ไบท์ โดยมีจุดศูนย์กลางอยู่ที่คำสั่งต่อจาก SJMP {-128 TO+127}

- JMP @A+DPTR ใช้รีจิสเตอร์ A เป็นตัว OFFSET {0-255} โดยการบวกค่าในรีจิสเตอร์ A กับ DPTR

CONDITIONAL JUMP

เป็นการกระโดดแบบมีเงื่อนไขโดยระยะทางในการกระโดดอยู่ในช่วง ๒๕๖ ไบท์ โดยเดินหน้าและถอยหลังจากจุดศูนย์กลางอยู่ที่คำสั่งที่ต่อจากการกระโดดนั้น {-128 TO+127}

- JZ กระโดดถ้าแอคคิวมูเลเตอร์เป็นศูนย์
- JNZ กระโดดถ้าแอคคิวมูเลเตอร์ไม่เท่ากับศูนย์
- JC กระโดดถ้าแฟลกตัวทดถูกเซ็ท
- JNC กระโดยถ้าแฟลกตัวทดไม่ถูกเซ็ท
- JB กระโดดถ้าบิทที่กำหนดถูกเซ็ท
- JNB กระโดดถ้าบิทที่กำหนดไม่ถูกเซ็ท
- JBC กระโดดถ้าบิทที่กำหนดถูกเซ็ทและทำการเคลียร์บิทที่กำหนดนั้นด้วย
- CJNE ทำการเปรียบเทียบโอเปอร์แรนด์ตัวแรกกับตัวที่ ๒ และจะกระโดยถ้าโอเปอร์แรนด์ทั้งสองมีค่าไม่เท่ากัน แฟลก CY จะถูกเซ็ทถ้าโอเปอร์แรนด์ตัวแรกมีค่าน้อยกว่าตัวที่ ๒
- DJNZ ลดค่าในโอเปอร์แรนด์และกระโดดถ้าผลของการลดค่านั้นไม่เท่ากันศูนย์

INTERUPT RETURNS

- RETI มีการทำงานเหมือนคำสั่ง RET แต่คำสั่ง RETI จะเพิ่มการ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ENABLE อินเตอร์รัพท์ด้วย

ในการดูคำสั่งของ ๘๐๓๑ จะมีสัญญาลักษณ์ที่ใช้ประกอบร่วมกันคำสั่งดังต่อไปนี้

Rn - รีจิสเตอร์ R0-R7 ใน BANK ที่กำหนดไว้

DIRECT - ตำแหน่งของหน่วยความจำ {RAM} ภายใน {0-127} หรือ I/O พอร์ตรีจิสเตอร์ควบคุม ฯลฯ

@RI - ตำแหน่งของ RAM {0-225} ซึ่งถูกกำหนดโดยค่าของ R1 หรือ RO

#DATA - ค่าคงที่ขนาด ๘ บิท

#DATA 16- ค่าคงที่ขนาด ๑๖ บิท

addr 16 - ตำแหน่งปลายทางขนาด ๑๖ บิท ใช้โดยคำสั่ง LCALL &LJMP

addr 11 - ตำแหน่งปลายทมงขนาด ๑๑ บิท ใช้โดยคำสั่ง ACALL & AJMP ซึ่งกระโดคได้ภายใน 2K ไบท์

rel - ค่าสัมพันธ์ {RELATIVE} โดยจะเป็นกำจวนที่มีเครื่องหมาย {-128 ถึง +127} ซึ่งจะสัมพันธ์กับตำแหน่งแรกของคำสั่งถัดไป

bit - ตำแหน่งของหน่วยความจำภายใน {RAM} ที่กำหนดได้โดยตรงหรือตำแหน่งในบิทต่าง ๆ ของ SFR

TIMER/COUNTER

๘๐๓๑ มี TIMER/COUNTER อยู่ ๒ ตัว คือ TO และ T1 สัญญาณ INPUT ที่จะป้อนให้ COUNTER นั้นทำงานที่ขอบขาลง {1 TO 0} คือ ต้องเป็นพัลส์ HIGH 1 แมนชีนไซเคิลและเป็น LOW 1 แมชีนไซเคิล ฉะนั้นความถี่สูงสุดที่ COUNTER นะนับได้นั้นประมาณ ๑/๒๔ ของความถี่ออสซิเลเตอร์ การทำงานของ TIMER/COUNTER แบ่งเป็น ๓ โหมด ดังกล่าวต่อไป

โหมด ๐

การทำงานในโหมดนี้รีจิสเตอร์ถูกกำหนดให้เป็นแบบ ๑๓ บิท โดยการนับจากค่าที่ทุกบิทเป็น HIGH ไปจนทุก ๆ บิทเป็น ๐ เกิด OVERFLOW และจะให้สัญญาณอินเตอร์รัพท์ โดยเซ็ทแฟลก TFO หรือ TF1 การให้จะให้ TIMER/COUNTER ตัวใดอยู่ในโหมด

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ใดนั้นกำหนดได้จากรีจิสเตอร์ TMOD

TMOD: TIMER/COUNTER MODE CONTROL REGISTOR

{MSB} {LSB}

GATF	C/T	M1	MO	GATE	C/T	M1	MO
TIMER 1				TIMER 0			

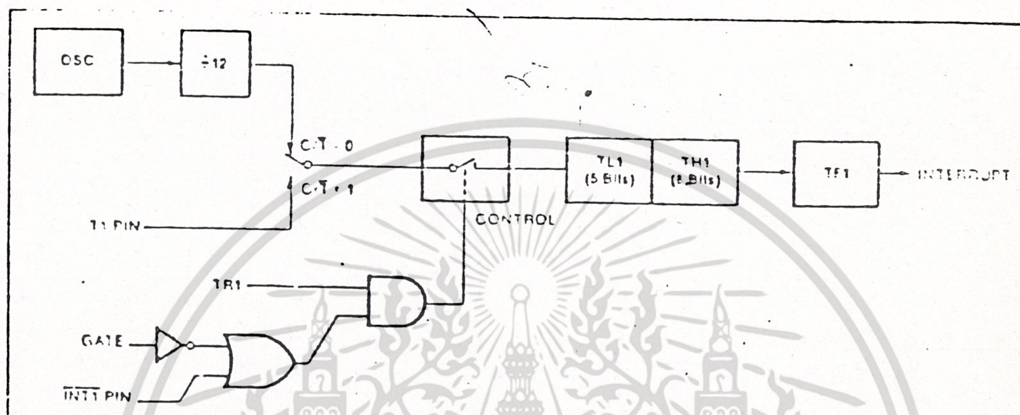
GATE GATING CONTROL WHEN SET TIMFR/COUNTER "X" IS ENABLE ONLY WHILE "INTx" PIN IS SET WHEN CLEARED TIMER "X" IS ENABLED WHENEVER "TRx" CONTROL BIT IS SET

C/T TIMER OR COUNTER SELECTOR CLEAR FOR TIME OPERATION {INPUT FROM INTERNAL SYSTER CLOCK}SET FOR COUNTER ORERATION {INPUT FROM "TX" INPUT PIN}

M1	M0	OPERATION MODE
0	0	MCS-48 TIMER "TLX" SERVES AS FIVE BIT PRESCALER
0	1	16 BIT TIMER/COUNTER "THx" AND "TLX" ARE CADCADED THERE IS NO PRESCALE
1	0	8 BIT AUTO-RELOAD TIMER/COUNTER 'THx" HOLDS A VALUE WHICH IS TO BE RELOADED INTO TLx EACH TIME IT OVERFLOWS
1	1	{TIMER0}TLO IS ANEIGHT BITTIMER/COUNTER CONTROLED BY STANDARD TIMER O CONTROL BITS THO IS THE EIGHT BIT TIMER ONLY CONTROLED

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

COUNTER จะทำงานได้ก็ต่อเมื่อ TR1 =1 และ GATE = 0 หรือ INT1 = 1 (ถ้าเซ็ท GATE = 1 TIMER/COUNTER จะถูกควบคุมด้วยสัญญาณ INT 1 จากภายนอก ประโยชน์ในการทำงานแบบนี้คือ ใช้วัดความกว้างของพัลส์จากอินพุทภายนอก TR1 เป็นบิทควบคุมอยู่ใน TCON ดังในรูป



รูป๘ TINER/COUNTER 1 MQDE 0 : 13 BIT COUNTER

(MSB)							(LSB)
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Symbol	Position	Name and Significance	Symbol	Position	Name and Significance
TF1	TCON.7	Timer 1 overflow Flag. Set by hardware on timer/counter overflow. Cleared by hardware when processor vectors to interrupt routine.	IE1	TCON.3	Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
TR1	TCON.6	Timer 1 Run control bit. Set/cleared by software to turn timer/counter on/off.	IT1	TCON.2	Interrupt 1 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.
TF0	TCON.5	Timer 0 overflow Flag. Set by hardware on timer/counter overflow. Cleared by hardware when processor vectors to interrupt routine.	IE0	TCON.1	Interrupt 0 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
TR0	TCON.4	Timer 0 Run control bit. Set/cleared by software to turn timer/counter on/off.	IT0	TCON.0	Interrupt 0 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.

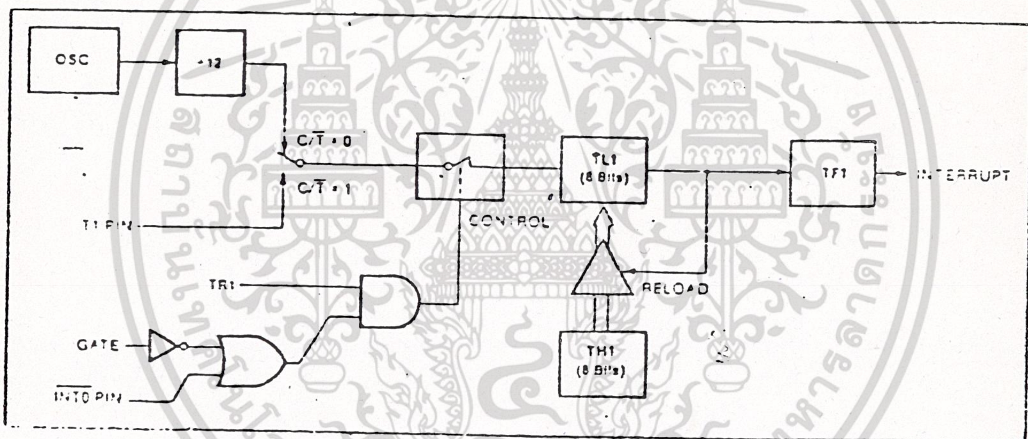
รูป ๙ TCON : TIMER/COUNTER CONTROL REGISTOR

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ในโหมด ๐ ที่จะแบ่ง TH1 เป็น ๘ บิท กับ TL1 อีก ๕ บิท โดยที่เหลืออีก ๓ บิท นั้นไม่ได้ใช้และการใช้งานจะเหมือนกันทั้ง TIMER 1 และ TIMER 0

โหมด ๑

ในโหมด ๒ รีจิสเตอร์จะเป็นแบบ ๘ บิท โดยที่ TL1 จะสามารถโหมดข้อมูลจาก TH1 ได้ใหม่ {AUTO-RELOAD} เมื่อเกิดโอเวอร์โฟลจาก TL1 (ดูรูป) โดยที่ค่าใน TH1 จะไม่ถูกเปลี่ยนการทำงานอื่น ๆ จะเหมือนกับโหมด ๐



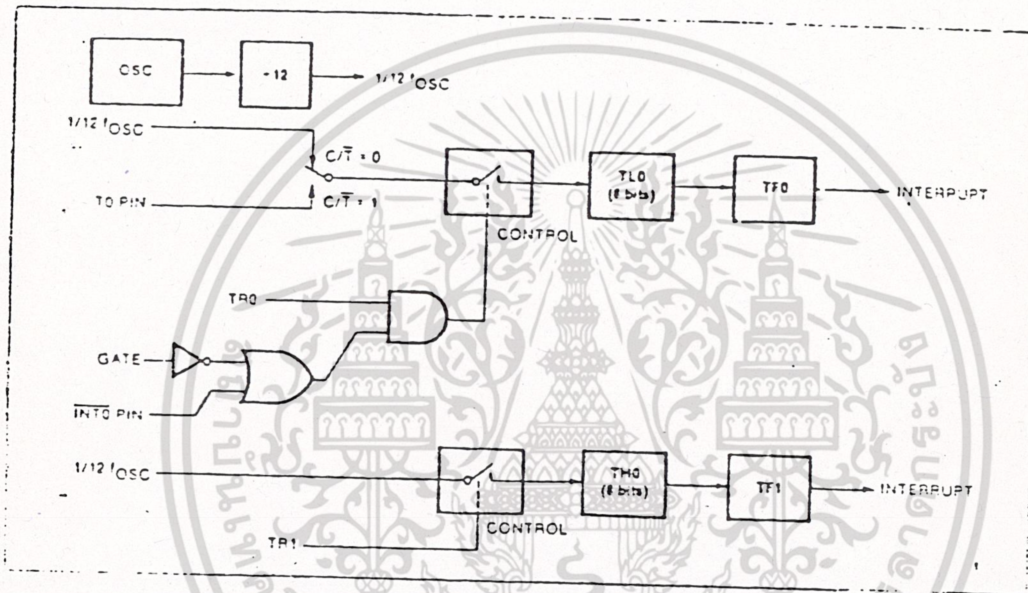
รูป๑๐TIMER/COUNTER 1 MODE 2 : 8 BIT AUTO-RELOAD

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

โหมด ๓

ในโหมด ๓ นี้จะแยก TLO และ THO ของ TIMER 0 ใช้โดยอิสระ TLO จะใช้บิทควบคุมคือ GATE, TRO, INTO และ TFO ส่วน THO ถูกใช้เป็น TIMER (นับแมชีนไซเคิล) และรับช่วงการใช้ TR1 และ TF1 ของ TIMER 1 ฉะนั้นในโหมด ๓ นี้ THO จะควบคุมการอินเตอร์รัพท์ของ TIMER 1 {TF1}

เมื่อใช้ TIMER 0 ในโหมด ๓แล้ว TIMER 1 สามารถจะสลับใช้ระหว่างโหมด ๓ และโหมดอื่นได้ หรือใช้เป็น BAUD RATE GENERATER



รูป ๑๑TIMER/COUNTER 0 MODE 3 : TWO 8 BIT COUNTERS

พอร์ตอนุกรม

พอร์ตอนุกรมนี้เป็น FULL DUPLEX ที่สามารถรับข้อมูลใน BYTE ที่สองได้โดยที่ BYIE แรกยังไม่ถูกอ่านออกไปจาก BUFFER แต่อย่างไรก็ตามข้อมูล BYTE แรกจะต้องถูกอ่านไปก่นอที่การรับข้อมูลใน BYTE ที่สองจะเสร็จสมบูรณ์มิฉะนั้นข้อมูล BYTE แรกสูญเสียไป (ถูกทับด้วยข้อมูลที่ตามมา) ข้อมูลที่จะใช้ในการส่งและรับจะถูกพักไว้ ณ ที่เดียวกันคือ SBUF

การเขียนข้อมูล ไปที่ SBUF จะเป็นการโหลดข้อมูลให้กับ TRANSMIT REGISTOR และการอ่าน SBUF จะเป็นการอ่านข้อมูลจาก RECEIVER REGISTOR พอร์ตอนุกรมแบ่งการทำงานออกเป็น ๔ โหมด

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

โหมด ๐ : ข้อมูลจะเข้ามาทาง RXD ส่วนข้อมูลทางออกจะออกทาง TXD ความเร็วในการส่ง {BAUD RATE} จะถูกกำหนดตามตัวเป็น ๑/๑๒ ของความถี่ออสซิลเลเตอร์ของระบบ ในโหมด ๐ จะเป็นการส่งข้อมูลขนาด ๘ บิท (โดย LSE ออกไปก่อน)

โหมด ๑ : ส่งและรับข้อมูลขนาด ๑๐ บิท ซึ่งประกอบด้วย START BIT {0}, ข้อมูล ๘ บิท {LSB ออกก่อน}, STOP BIT ในขณะรับข้อมูล STOP BIT จะถูกส่งให้ RB8 ในรีจิสเตอร์หน้าที่พิเศษ SCON ความเร็วในการส่งไม่กำหนดตายตัว (ดูการหา BAUD RATE)

โหมด ๒ : ส่งและรับข้อมูลขนาด ๑๑ บิท ประกอบด้วย START BIT {0}, ข้อมูล ๘ บิท {LSB ก่อน} , ข้อมูลบิทที่ ๙ ที่สามารถโปรแกรมได้ และอีก ๑ STOP BIT บิทที่ ๙ ของข้อมูลสามารถ SET เป็น ๐ หรือ ๑ ก็ได้ประโยชน์อาจใช้เป็นตัวส่งพาริตี้บิทโดยนำค่าของแฟลก P ใน PSW มาไว้ใน TB8 ของ SCON ความเร็วในการส่งจะถูกโปรแกรมเป็น ๑/๓๒ หรือ ๑/๖๔ ของออสซิลเลเตอร์

โหมด ๓ : การทำงานเหมือนกับโหมด ๒ เพียงแต่ความเร็วในการส่งไม่กำหนดตายตัว

การทำงานทั้ง ๔ โหมด ทางด้านส่งจะเริ่มการส่งขึ้นก็ต่อเมื่อ SBUF ถูกใช้เป็นปลายทางของคำสั่งต่าง ๆ เช่น MOV SEUF,A ในการด้านรับการรับจะเริ่มก็ต่อเมื่อ RI=O และ REN = 1 ในโหมด ๐ ส่วนโหมดอื่น ๆ การรับเข้ามูลจะเริ่มต้นเมื่อมี START BIT เข้ามา และREN=1

MULTIPROCESSOR COMMUNICATIONS

ในโหมด ๒ และ ๓ มีเงื่อนไขพิเศษสำหรับการติดต่อระหว่างหน่วยประมวลผลหลายตัวในโหมดนี้ข้อมูลในบิทที่ ๙ จะถูกโหลดเข้าไปใน RB8 ต่อจากนั้นจึงจะรับ STOP BIT อินเตอร์รัพท์แฟลกของพอร์ตอนุกรมแอคทีฟก็ต่อเมื่อ RB8 = 1 การจะใช้ลักษณะพิเศษนี้ต้องเซ็ท SM2 ใน SCON

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เมื่อตัวประมูลผลตัวแม่ต้องการจะส่งข้อมูลเป็นบล๊อกให้กับตัวลูกซึ่งมีอยู่หลายตัว ฉะนั้น BYTE แรกที่จะต้องส่งคือ ตัวกำหนดว่าจะให้ตัวใดเป็นตัวรับ (การกำหนด ADDRESS) โดยที่ BYTE ที่เป็น ADDRESS จะต้องแตกต่างจาก BYTE ข้อมูลซึ่งเราทำได้โดยให้บิทที่ ๙ ของข้อมูลเป็น ๑ ใน BYTE ที่กำหนดให้เป็น ADDRESS ของตัวลูก {SLAVE} ส่วนในตัวลูกทุกตัวต้องเซ็ท SM2 = 1 เมื่อทำดังนี้หน่วยประมวลผลที่เป็นตัวลูกทุกตัวจะไม่ถูกอินเตอร์รัพท์ที่เกิดจากการส่งข้อมูลอนุกรมถ้าข้อมูลที่ส่งนั้นบิทที่ ๙ ไม่ถูกเซ็ท ฉะนั้นหน่วยประมวลผลตัวลูกทุกตัวจะรับการส่งที่กำหนดให้เป็น ADDRESS แต่จะมีเพียงตัวเดียวที่ตรวจสอบแล้วว่าถูกกำหนดให้เป็นปลายทางในการส่งข้อมูลจึงเตรียมการรับบล๊อกของข้อมูลและเคลียร์ SM 2 ด้วย ส่วนตัวอื่น ๆ คงปล่อยให้ SM2 เซ็ทอยู่อย่างเดิมและทำงานจามปกติต่อไป

SM2 นี้จะไม่มีผลในการใช้งานในโหมด ๐ ส่วนในโหมด ๑ จะใช้สำหรับตรวจสอบความถูกต้องของ STOP BIT ในโหมด ๑ เมื่อเซ็ท SM2 = 1 และ RI {RECIEVE INTERURT} จะไม่ถูก SET ถ้าไม่ได้รับ STOP BIT ที่ถูกต้อง

SERIAL PORT CONTROL REGISTOR

ในรูป เป็นรายละเอียดของ SFR ที่ทำหน้าที่ควบคุมพอร์ตอนุกรม {SCON} ภายใน SCON ไม่เพียงแต่มีการเลือกโหมดเท่านั้นแต่ยังรวมทั้งบิทที่ ๙ ของข้อมูล {TB8,RB8} และ SERIAL PORT INTERUPT {T1 และ R1}

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0 SCON. 7 Serial Port mode specifier. (NOTE 1).

SM1 SCON. 6 Serial Port mode specifier. (NOTE 1).

SM2 SCON. 5 Enables the multiprocessor communication feature in modes 2 & 3. In mode 2 or 3, if SM2 is set to 1 then RI will not be activated if the received 9th data bit (RB8) is 0. In mode 1, if SM2 = 1 then RI will not be activated if a valid stop bit was not received. In mode 0, SM2 should be 0. (See Table 9)

REN SCON. 4 Set/Cleared by software to Enable/Disable reception.

TB8 SCON. 3 The 9th bit that will be transmitted in modes 2 & 3. Set/Cleared by software.

RB8 SCON. 2 In modes 2 & 3, is the 9th data bit that was received. In mode 1, if SM2 = 0, RB8 is the stop bit that was received. In mode 0, RB8 is not used.

TI SCON. 1 Transmit interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or at the beginning of the stop bit in the other modes. Must be cleared by software.

RI SCON. 0 Receive interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or halfway through the stop bit time in the other modes (except see SM2). Must be cleared by software.

NOTE 1:

SM0	SM1	Mode	Description	Baud Rate
0	0	0	SHIFT REGISTER	Fosc./12
0	1	1	8-Bit UART	Variable
1	0	2	9-Bit UART	Fosc./64 OR Fosc./32
1	1	3	9-Bit UART	Variable

รูป ๑๒ SCON : SERIAL PORT CONTROL REGISTOR

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

อัตราความเร็วในการส่ง {BAUD RATES}

ในโหมด ๐ ความเร็วในการส่งกำหนดไว้แน่นอนคือ

$$\text{MODE 0 BAUD RATE} = \frac{\text{OSCILLATOR FREQUENCY}}{12}$$

ในโหมด ๒ ความเร็วในการส่งขึ้นอยู่กับ SMOD ซึ่งอยู่ใน PCON ถ้า SMOD=0 ความเร็วจะเท่ากับ ๑/๖๔ ของความถี่ออสซิลเลเตอร์ ถ้า SMOD = 1 ความเร็วจะเป็น ๑/๓๒ ของความถี่ออสซิลเลเตอร์

ความเร็วในการส่งในโหมด ๒ คำนวณได้จากสูตรดังนี้

MODE 2 BAUD RATE = 2 X {OSCILLATOR FREQUENCY}

โหมด ๑ และโหมด ๓ ความเร็วในการส่งถูกกำหนดโดยอัตราของ OVERFLOW ของ TIMER 1

การใช้ TIMER 1 ในการกำเนิด BAUD RATE

เมื่อใช้ TIMER 1 เป็นตัวกำหนด BAUD RATE ความเร็วจะขึ้นอยู่กับ OVERFLOW RATE และค่าที่อยู่ใน SMOD ความเร็วคำนวณได้จากสูตร

MODE 1,3 BAUD RATE = 2 X {TIMER 1 OVERFOLW RATE}

ในการใช้ TIMER 1 เป็นตัวกำเนิดความเร็วในการส่งข้อมูลนี้จะต้องไม่ยอมให้มีการอินเตอร์รัพท์ของ TIMER 1 วิธีการใช้ TIMER 1 เป็นตัวกำเนิดความเร็วนี้โดยทั่วไปเราจะเซ็ทให้ TIMER 1/COUNTER เป็น TIMER และอยู่ในโหมด ๒ ซึ่ง TIMER ในโหมด ๒ นี้ทำ AUTO-RELOAD ได้ (เซ็ทไบท์สูงของ TMOD =001OB) ในกรณีนี้ความเร็วจะคำนวนได้จากสูตร

MODE 1,3 BAUD RATE = 2 X OSCILLATOR FREQUENCY

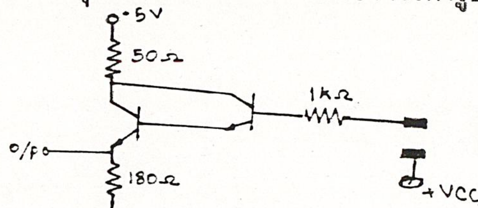
โดยที่ค่า TH1 จะเป็นค่าในช่อง RELOAD ของตารางดังนี้

BAUD RATE	FOSC	SMOD	TIMER 1		
			C/T	MODE	RELOAD VALUE
MODE 0 MAX : 1MHX					

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การทดลอง

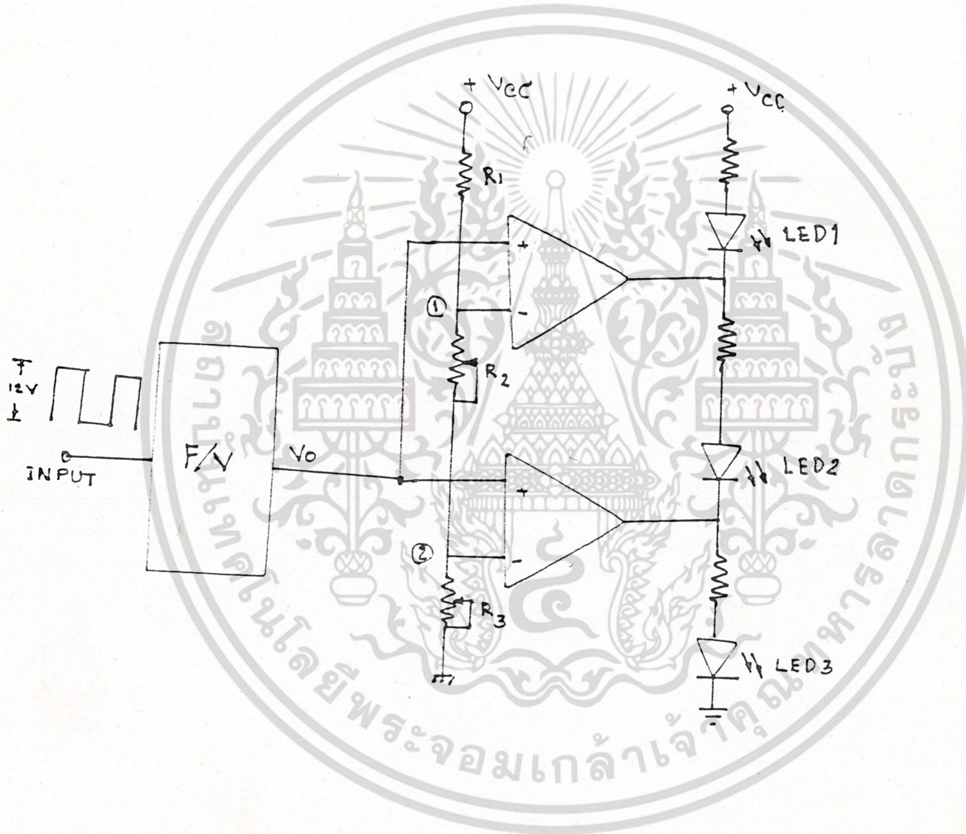
๑) ทดลองวงจรชุด Sensor น้ำฝน โดยต่อวงจรดังรูป



ถ้าชุด Sensor โดนน้ำจะทำให้ output ได้ระดับ Logic = 1

" " " = 0

๒) ทำการทดลองและดูผลของวงจรวัดความเร็วรอบ



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การทดลอง

๑. สมมติให้ความเร็วรอบที่ให้แรงบิดสูงสุดเท่ากับ ๓๘๐๐-๔๕๐๐ รอบ

๒. หาความถี่ i/p ที่ ๓๘๐๐ จากสูตร

$$f = n/30 = 3800/30 = 126.6 \text{ Hz}$$

๓. หาความถี่ที่ ๔๕๐๐ รอบ

$$f = 4500/30 = 150 \text{ Hz}$$

๔.ป้อนพัลซ์ความถี่ ๑๒๖.๖ Hz วัด Vo อ่านค่าไว้ สมมติ = Va

๕.ป้อนพัลซ์ความถี่ ๑๕๐ Hz อ่านค่าของไว้ สมมติ = Vb

๖.ทำ Voltage reference โดยการปรับ R2 และ R3 ให้ Voltage ที่จุด ๒ เท่ากับ Va และให้ Voltage ที่จุด ๑ เท่ากับ Vb

๗.ทำซ้ำขั้นตอนที่ ๔ ถึง ๖ อีกครั้งเพื่อความแน่นอน

หมายเหตุ พัลซ์ที่ป้อนให้กับอินพุทต้องมีแอมปลิจูดเท่ากับ ๑๒ V เสมอและถ้านำไปใช้กับเครื่องยนต์ที่ให้แรงบิดสูงสุดที่ความเร็วรอบต่างจากนี้จะต้องคำนวนหาความถี่ใหม่

๓) ทำการต่อวงจร Inter face กับ Microcontroller ทดลองวงจร input port และ out put port โดยใช้โปรแกรมดังนี้

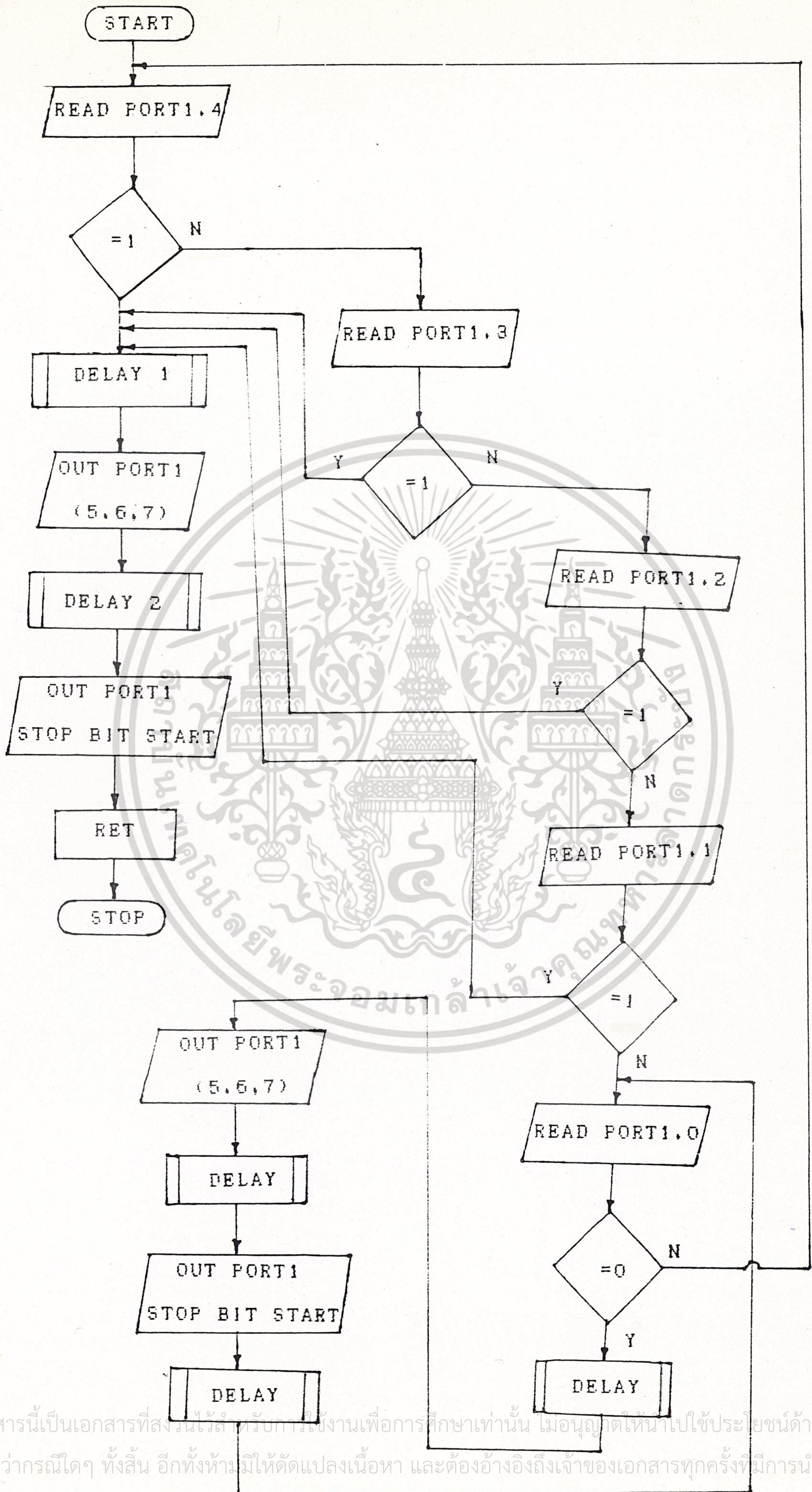
```
        mov     a,#ffH
        mov     p1,a
d1:     mov     r0,#ffH
d2:     nop
        mov     r1,#ffH
d3:     nop
        djnz    r1,d3
        djnz    r0,d2
```

ทำอย่างนี้จนครบทุก port โดยเปลี่ยนค่าเบอร์ port แล้วทำการนำเอา logic probe วัดดูที่ขาต่าง ๆ จนครบ

๔) จากนั้นนำเอาวงจรทั้งหมดมาทำการ interface กับด้วยกันแล้วทดลองเขียนโปแกรมใช้งานจริง ๆ ดู ดูผลการทำงานของ Relay แต่ละตัว

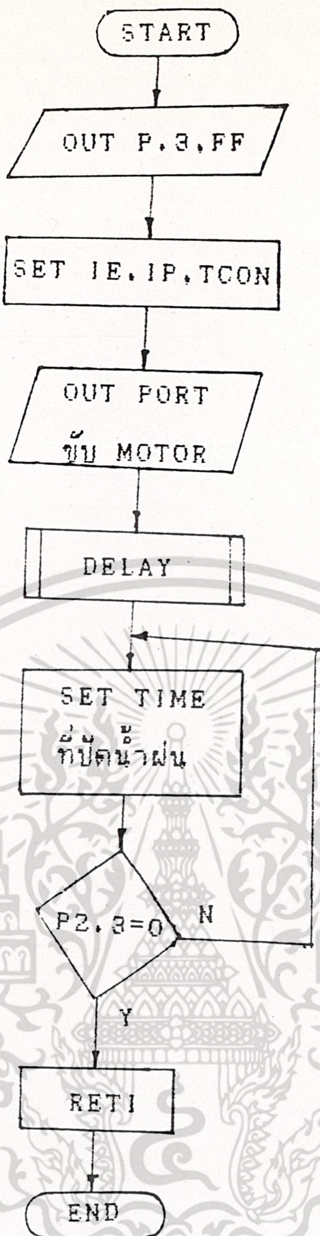
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ประตูและ LOCK

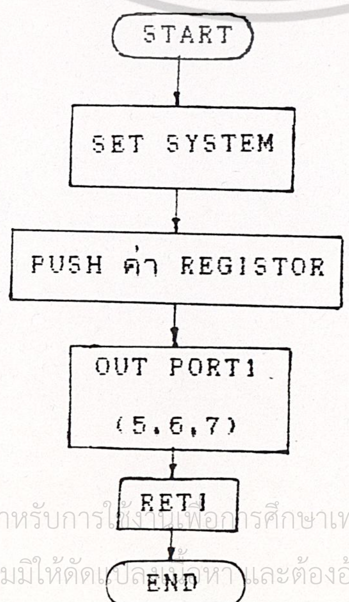


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

SENSOR น้ำฝน (INTO)



เครื่องวัดความเร็วรอบ



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
;@@@@@@@automobile  controller  program@@@@@@@@
                org   00h
                ljmp  begin
;*****section interrup0 program*****
                org   03h
                ajmp  service_int0              ;go service interrup0
;*****section door and locker program******
                org    100h
        begin:  clr    p1.5
                mov    sp,#10h
                mov    ie,#10000001b      ;set ie register
                mov    tcon,#00000010b    ;set tcon   register
                mov    ip,#00000001b      ;set ip   register
                nop
      check_1:  jb     p3.3,door1
                acall  safety_1
        door1:  jb     p1.4,door2
                acall  gen_sound1
        door2:  jb     p1.3,door3
                acall  gen_sound1
        door3:  jb     p1.2,door4
                acall  gen_sound1
        door4:  jb     p1.1,lock
                acall  gen_sound1
         lock:  jb     p1.0,check_1
                acall  gen_sound2
                sjmp   lock
   gen_sound1:  mov    r0,#0dh
       delay1:  nop
                mov    r1,#0ffh
       delay2:  nop
                mov    r2,#0ffh
       delay3:  nop
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        djnz    r2,delay3
        djnz    r1,delay2
        djnz    r0,delay1
        mov     a,#3fh              ;set   channal1
        mov     p1,a                ;out   channal1
        mov     r3,#0ffh
    l1: nop
        mov     r4,#0ffh
    l2: nop
        djnz    r4,l2
        djnz    r3,l1
        clr     p1.5                ;stop channal1
        mov     r5,#0dh
    l3: nop
        mov     r6,#0ffh
    l4: nop
        mov     r7,#0ffh
    l5: nop
        djnz    r7,l5
        djnz    r6,l4
        djnz    r5,l3
        ret
gen_sound2:  mov     r0,#12h
     delay5: nop
             mov     r1,#0ffh
     delay6: nop
             mov     r2,#0ffh
     delay7: nop
             djnz    r2,delay7
             djnz    r1,delay6
             djnz    r0,delay5
             mov     a,#0bfh        ;set   channal2
             mov     p1,a           ;out   channal2
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        mov     r3,#0ffh
d1:     nop
        mov     r4,#0ffh
d2:     nop
        djnz    r4,d2
        djnz    r3,d1
        clr     p1.5                    ;stop channal2
        ret
;****************************************
;    subrutine   int0   program
;****************************************

service_int0:
        push    acc
        mov     psw,#00001000b
        mov     a,#00h
        mov     p3,a                    ;drive motor
        mov     r4,#15h
delay9: nop
        mov     r5,#0ffh
delay10: nop
        mov     r6,#0ffh
delay11: nop
        djnz    r6,delay11
        djnz    r5,delay10
        djnz    r4,delay9
        mov     a,#0fah                 ;stop work  glass
        mov     p3,a                    ;sensor stop bit
        mov     r7,#0ffh
loop1:  djnz    r7,loop1
loop2:  setb    p3.0                    ;motor1 stop
        mov     r0,#0dh
delay12: nop
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
            mov     r1,#0ffh
delay13:    nop
            mov     r2,#0ffh
delay14:    nop
            djnz    r2,delay14
            djnz    r1,delay13
            djnz    r0,delay12
            clr     p3.0                ;motor1 working
            mov     r3,#1ch             ;delay time of motor1
delay15:    nop
            mov     r4,#0ffh
delay16:    nop
            mov     r5,#0ffh
delay17:    nop
            djnz    r5,delay17
            djnz    r4,delay16
            djnz    r3,delay15
            jb      p3.7,loop2
            setb    p3.0
            setb    p3.2
            anl     psw,#00h
            pop     acc
            mov     ie,#10000001b
            reti
safety_1:   push    acc
            mov     psw,#00011000b
R_1:        mov     r1,#12h
t3:         nop
            mov     r2,#0ffh
t4:         nop
            mov     r3,#0ffh
t5:         nop
            djnz    r3,t5
```

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                djnz    r2,t4
                djnz    r1,t3
                mov     a,#7fh
                mov     p1,a
                mov     r6,#0ffh
t6:             nop
                mov     r7,#0ffh
t7:             nop
                djnz    r7,t7
                djnz    r6,t6
                clr     p1.5
                jnb     p3.3,R_1
                anl     psw,#00h
                pop     acc
                ret
                end
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สรุปผลและวิจารณ์ผลการทดลอง

เนื่องจาก Mcs-51 (๘๐๓๑) เป็น Microcontroller แบบ Single Chip ฉะนั้นการ interfacing กับอุปกรณ์ภายนอกก็สามารถใช้ latch input และ latch output ได้เลยจากการทดลองชุดที่เป็น digital ไม่ค่อยมีปัญหาแต่จะมีปัญหาตอนการเขียนโปรแกรมหน่วงเวลการทำงานกับอุปกรณ์ภายนอกจึงต้องใช้ความละเอียด คือให้อุปกรณ์ภายนอกทำงานให้ทันกับ microcontrolle

เนื่องจากเครื่องวัดความเร็วรอบนี้ทำหน้าที่วัดความเร็วรอบของเครื่องยนต์โดยปกติในรถยนต์ทั่ว ๆ ไป เครืองจัดความเร็วรอบเป็น จะเป็นระบบเข็มมิเตอร์ซื่อมีส่วนประกอบภายในที่ละเอียดอ่อนและบอบบางมาก ถ้านำไปใช้กับรถยนต์ที่มีการกระแทกระเทือนตลอดเวลา ทำให้อายุการใช้งานของเครื่องวัดแบบเข็มสั้นลง และทำมให้อ่านค่าผิดพลาดไปด้วย ปัญหานี้สารถแก้ไขได้ด้วยเครื่องวัดความเร็วรอบแบบแสดงผลเป็น led และยังมีคุณสมบัติพิเศษอีกอย่างหนึ่งคือ เมื่อความเร็วรอบสูงเกิดไป วงจรนี้จะส่งเสัญญาณไปให้วรจรเสียงพูดทำงาน พูดบอกกับเราได้โดยตรงเลย

วงจรจะแบ่งออกเป็น ๓ ภาคคือ

๑.ภาควัดความถี่และเปลี่ยนความถี่เป็นแรงดัน

๒.ภาคแสดงผลโดยใช้ led แสงระดับของแรงดันพร้อมทั้งส่งสัญญาณไฟเสียงพูด

๓.ภาคควบคุมแรงด้านให้คงที่และแรงดันอ้างอิง

ภาคเปลี่ยความถี่เป็นแรงดันไฟตรง จะแปลงความถี่จากหน้าทองขาวในจานจ่าย ถ้าความถี่มาก (ความเร็วของเครื่องยนต์สูง) แรงดันก็จะมากตามแรงดันนี้ จะถูกป้อนเข้าวงจรเปรียบเทียบ ๓ ชุด ซึ่งจะทำหน้าที่ขับ led ในชุดของมัน

การปรับแต่ง

ก่อนอื่นจะต้องรู้ข้อมูลต่าง ๆ ของเครื่องยนต์นั้น ๆ เสียก่อนดังนี้คือ

๑.ความเร็วรอบที่เครื่องยนต์ให้แรงบิดสูงสุด

๒.เครื่องยนต์ต้องเป็นเครืองยนต์ ๔ สูบ ๔ จังหวะ

๓.คำนวนหารความถี่ที่ป้อนให้ input ที่ความเร็วรอบต่าง ๆ จากสูตร

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

$f = n / 30$

; f = ความถี่ {Hz}

n = จำนวนรอบ {rpm}

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ภาคผนวก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

MCS®-51 INSTRUCTION SET

Table 10. 8051 Instruction Set Summary

Interrupt Response Time: Refer to Hardware Description Chapter.

Instructions that Affect Flag Settings[1]

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

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

Note on instruction set and addressing modes:

- Rn — Register R7–R0 of the currently selected Register Bank.
- direct — 8-bit internal data location's address. This could be an Internal Data RAM location (0–127) or a SFR [i.e., I/O port, control register, status register, etc. (128–255)].
- @Ri — 8-bit internal data RAM location (0–255) addressed indirectly through register R1 or R0.
- # data — 8-bit constant included in instruction.
- # data 16 — 16-bit constant included in instruction.
- addr 16 — 16-bit destination address. Used by LCALL & LJMP. A branch can be anywhere within the 64K-byte Program Memory address space.
- addr 11 — 11-bit destination address. Used by ACALL & AJMP. The branch will be within the same 2K-byte page of program memory as the first byte of the following instruction.
- rel — Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is −128 to +127 bytes relative to first byte of the following instruction.
- bit — Direct Addressed bit in Internal Data RAM or Special Function Register.

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

All mnemonics copyrighted ©Intel Corporation 1980

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Table 10. 8051 Instruction Set Summary (Continued)

Mnemonic		Description	Byte	Oscillator Period
ARITHMETIC OPERATIONS (Continued)				
INC	DPTR	Increment Data Pointer	1	24
MUL	AB	Multiply A & B	1	48
DIV	AB	Divide A by B	1	48
DA	A	Decimal Adjust Accumulator	1	12
LOGICAL OPERATIONS				
ANL	A,Rn	AND Register to Accumulator	1	12
ANL	A,direct	AND direct byte to Accumulator	2	12
ANL	A,@Ri	AND indirect RAM to Accumulator	1	12
ANL	A,#data	AND immediate data to Accumulator	2	12
ANL	direct,A	AND Accumulator to direct byte	2	12
ANL	direct,#data	AND immediate data to direct byte	3	24
ORL	A,Rn	OR register to Accumulator	1	12
ORL	A,direct	OR direct byte to Accumulator	2	12
ORL	A,@Ri	OR indirect RAM to Accumulator	1	12
ORL	A,#data	OR immediate data to Accumulator	2	12
ORL	direct,A	OR Accumulator to direct byte	2	12
ORL	direct,#data	OR immediate data to direct byte	3	24
XRL	A,Rn	Exclusive-OR register to Accumulator	1	12
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	12
XRL	A,@Ri	Exclusive-OR indirect RAM to Accumulator	1	12
XRL	A,#data	Exclusive-OR immediate data to Accumulator	2	12
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	12
XRL	direct,#data	Exclusive-OR immediate data to direct byte	3	24
CLR	A	Clear Accumulator	1	12
CPL	A	Complement Accumulator	1	12

Mnemonic		Description	Byte	Oscillator Period
LOGICAL OPERATIONS (Continued)				
RL	A	Rotate Accumulator Left	1	12
RLC	A	Rotate Accumulator Left through the Carry	1	12
RR	A	Rotate Accumulator Right	1	12
RRC	A	Rotate Accumulator Right through the Carry	1	12
SWAP	A	Swap nibbles within the Accumulator	1	12
DATA TRANSFER				
MOV	A,Rn	Move register to Accumulator	1	12
MOV	A,direct	Move direct byte to Accumulator	2	12
MOV	A,@Ri	Move indirect RAM to Accumulator	1	12
MOV	A,#data	Move immediate data to Accumulator	2	12
MOV	Rn,A	Move Accumulator to register	1	12
MOV	Rn,direct	Move direct byte to register	2	24
MOV	Rn,#data	Move immediate data to register	2	12
MOV	direct,A	Move Accumulator to direct byte	2	12
MOV	direct,Rn	Move register to direct byte	2	24
MOV	direct,direct	Move direct byte to direct	3	24
MOV	direct,@Ri	Move indirect RAM to direct byte	2	24
MOV	direct,#data	Move immediate data to direct byte	3	24
MOV	@Ri,A	Move Accumulator to indirect RAM	1	12

All mnemonics copyrighted © Intel Corporation 1980

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Table 10. 8051 Instruction Set Summary (Continued)

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

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

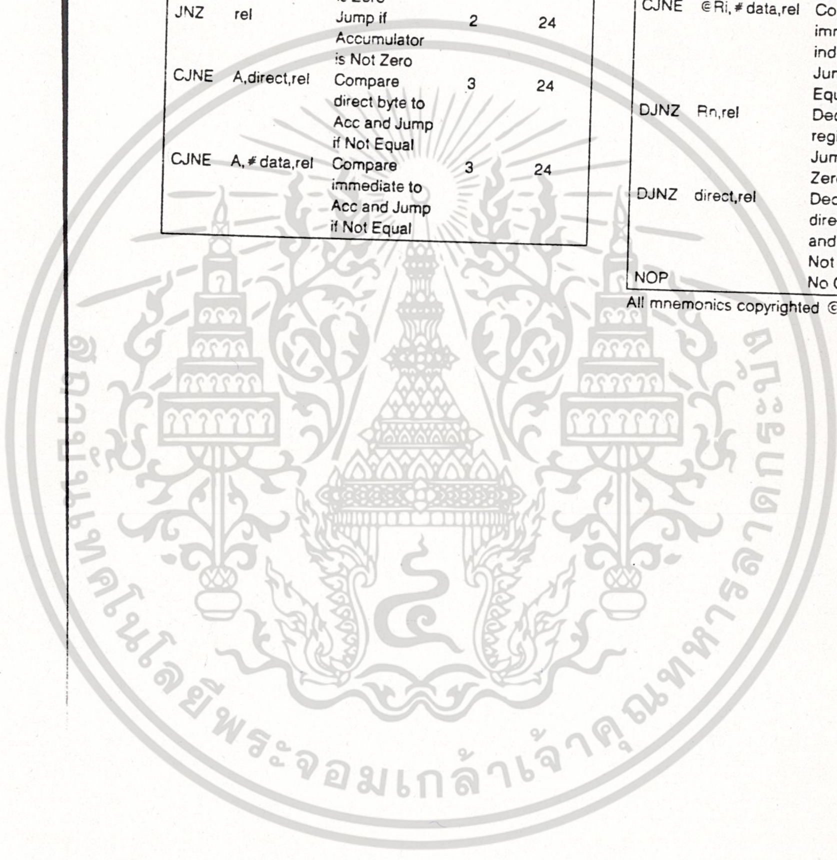
All mnemonics copyrighted © Intel Corporation 1980

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Table 10. 8051 Instruction Set Summary (Continued)

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

All mnemonics copyrighted © Intel Corporation 1980



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Table 11. Instruction Opcodes in Hexadecimal Order

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Table 11. Instruction Opcodes in Hexadecimal Order (Continued)

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Table 11. Instruction Opcodes in Hexadecimal Order (Continued)

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

INSTRUCTION DEFINITIONS

ACALL addr11

Function: Absolute Call

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

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

ACALL SUBRTN

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

Bytes: 2

Cycles: 2

Encoding:

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

Operation: ACALL
(PC) ← (PC) + 2
(SP) ← (SP) + 1
((SP)) ← (PC_{7-0})
(SP) ← (SP) + 1
((SP)) ← (PC_{15-8})
(PC_{10-0}) ← page address

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ADD A,<src-byte>

Function: Add

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

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

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

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

ADD A,R0

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

ADD A,Rn

Bytes: 1

Cycles: 1

Encoding: | 0 0 1 0 | 1 r r r |

Operation: ADD
(A) ← (A) + (Rn)

ADD A,direct

Bytes: 2

Cycles: 1

Encoding: | 0 0 1 0 | 0 1 0 1 | | direct address |

Operation: ADD
(A) ← (A) + (direct)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ADD A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 1 0	0 1 1 i

Operation: ADD
$(A) \leftarrow (A) + ((R_i))$

ADD A,#data

Bytes: 2

Cycles: 1

Encoding:

0 0 1 0	0 1 0 0	immediate data

Operation: ADD
$(A) \leftarrow (A) + \#data$

ADDC A,<src-byte>

Function: Add with Carry

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

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

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

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

```
ADDC   A,R0
```

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ADDC A,Rn

Bytes: 1

Cycles: 1

Encoding: | 0 0 1 1 | 1 r r r |

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

ADDC A,direct

Bytes: 2

Cycles: 1

Encoding: | 0 0 1 1 | 0 1 0 1 | direct address |

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

ADDC A,@Ri

Bytes: 1

Cycles: 1

Encoding: | 0 0 1 1 | 0 1 1 i |

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

ADDC A,#data

Bytes: 2

Cycles: 1

Encoding: | 0 0 1 1 | 0 1 0 0 | immediate data |

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

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

AJMP addr11

Function: Absolute Jump

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

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

AJMP JMPADR

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

Bytes: 2

Cycles: 2

Encoding:

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

Operation: AJMP
(PC) ← (PC) + 2
(PC_{10-0}) ← page address

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

Function: Logical-AND for byte variables

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

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

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

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

ANL A,R0

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

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

ANL P1,#01110011B

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ANL A,Rn

Bytes: 1

Cycles: 1

Encoding: | 0 1 0 1 | 1 r r r |

Operation: ANL
(A) ← (A) ∧ (Rn)

ANL A,direct

Bytes: 2

Cycles: 1

Encoding: | 0 1 0 1 | 0 1 0 1 | | direct address |

Operation: ANL

(A) ← (A) ∧ (direct)

ANL A,@Ri

Bytes: 1

Cycles: 1

Encoding: | 0 1 0 1 | 0 1 1 i |

Operation: ANL

(A) ← (A) ∧ ((Ri))

ANL A,#data

Bytes: 2

Cycles: 1

Encoding: | 0 1 0 1 | 0 1 0 0 | | immediate data |

Operation: ANL
(A) ← (A) ∧ #data

ANL direct,A

Bytes: 2

Cycles: 1

Encoding: | 0 1 0 1 | 0 0 1 0 | | direct address |

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ANL direct,#data

Bytes: 3

Cycles: 2

Encoding: | 0 1 0 1 | 0 0 1 1 | direct address | immediate data |

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

ANL C,<src-bit>

Function: Logical-AND for bit variables

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

Only direct addressing is allowed for the source operand.

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

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

ANL C,bit

Bytes: 2

Cycles: 2

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

Operation: ANL
(C) ← (C) ∧ (bit)

ANL C,/bit

Bytes: 2

Cycles: 2

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

Operation: ANL
(C) ← (C) ∧ ¬ (bit)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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

Function: Compare and Jump if Not Equal.

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

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

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

```
            CJNE   R7,#60H, NOT_EQ
;           ...    .....              ;  R7 = 60H.
NOT_EQ:     JC     REQ_LOW            ;  IF R7 < 60H.
;           ...    ....               ;  R7 > 60H.
```

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

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

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

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

CJNE A,direct,rel

Bytes: 3

Cycles: 2

Encoding: | 1 0 1 1 | 0 1 0 1 | direct address | rel. address |

Operation:

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

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


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

CJNE A,#data,rel

Bytes: 3

Cycles: 2

Encoding: | 1 0 1 1 | 0 1 0 0 | immediate data | rel. address |

Operation:
(PC) ← (PC) + 3
IF (A) < > *data*
THEN
(PC) ← (PC) + *relative offset*

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

CJNE Rn,#data,rel

Bytes: 3

Cycles: 2

Encoding: | 1 0 1 1 | 1 r r r | immediate data | rel. address |

Operation:
(PC) ← (PC) + 3
IF (Rn) < > *data*
THEN
(PC) ← (PC) + *relative offset*

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

CJNE @Ri,#data,rel

Bytes: 3

Cycles: 2

Encoding: | 1 0 1 1 | 0 1 1 i | immediate data | rel. address |

Operation:
(PC) ← (PC) + 3
IF ((Ri)) < > *data*
THEN
(PC) ← (PC) + *relative offset*

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

CLR A

Function: Clear Accumulator

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

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

CLR A

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

Bytes: 1

Cycles: 1

Encoding:

1 1 1 0	0 1 0 0

Operation: CLR
(A) ← 0

CLR bit

Function: Clear bit

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

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

CLR P1.2

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

CLR C

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	0 0 1 1

Operation: CLR
(C) ← 0

CLR bit

Bytes: 2

Cycles: 1

Encoding:

1 1 0 0	0 0 1 0	bit address

Operation: CLR
(bit) ← 0

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

CPL A

Function: Complement Accumulator

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

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

CPL A

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

Bytes: 1

Cycles: 1

Encoding: | 1 1 1 1 | 0 1 0 0 |

Operation: CPL
(A) ← ¬ (A)

CPL bit

Function: Complement bit

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

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

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

CPL P1.1

CPL P1.2

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

CPL C

Bytes: 1

Cycles: 1

Encoding: | 1 0 1 1 | 0 0 1 1 |

Operation: CPL
(C) ← ¬ (C)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

CPL bit

Bytes: 2

Cycles: 1

Encoding:

1 0 1 1	0 0 1 0		bit address

Operation: CPL
(bit) ← ¬ (bit)

DA A

Function: Decimal-adjust Accumulator for Addition

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

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

If the carry flag is now set, or if the four high-order bits now exceed nine (1010xxxx-111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this would set the carry flag if there was a carry-out of the high-order bits, but wouldn't clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

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

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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

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

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

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

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

```
ADD   A,#99H

DA    A
```

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

Bytes: 1

Cycles: 1

Encoding:

1 1 0 1	0 1 0 0

Operation: DA

-contents of Accumulator are BCD

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

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

AND

IF $[[(A_{7-4}) > 9] \vee [(C) = 1]]$

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

DEC byte

Function: Decrement

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

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

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

DEC @R0

DEC R0

DEC @R0

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

DEC A

Bytes: 1

Cycles: 1

Encoding: | 0 0 0 1 | 0 1 0 0 |

Operation: DEC
(A) ← (A) − 1

DEC Rn

Bytes: 1

Cycles: 1

Encoding: | 0 0 0 1 | 1 r r r |

Operation: DEC
(Rn) ← (Rn) − 1

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

DEC direct

Bytes: 2

Cycles: 1

Encoding: | 0 0 0 1 | 0 1 0 1 | | direct address |

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

DEC @Ri

Bytes: 1

Cycles: 1

Encoding: | 0 0 0 1 | 0 1 1 i |

Operation: DEC
((Ri)) ← ((Ri)) − 1

DIV AB

Function: Divide

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

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

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

DIV AB

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

Bytes: 1

Cycles: 4

Encoding: | 1 0 0 0 | 0 1 0 0 |

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

DJNZ <byte>,<rel-addr>

Function: Decrement and Jump if Not Zero

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

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

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

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

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

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

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

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

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

DJNZ Rn,rel

Bytes: 2

Cycles: 2

Encoding:

1 1 0 1	1 r r r		rel. address

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

DJNZ direct,rel

Bytes: 3

Cycles: 2

Encoding: | 1 1 0 1 | 0 1 0 1 | | direct address | | rel. address |

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

INC <byte>

Function: Increment

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

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

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

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

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

INC A

Bytes: 1

Cycles: 1

Encoding: | 0 0 0 0 | 0 1 0 0 |

Operation: INC
(A) ← (A) + 1

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

INC Rn

Bytes: 1

Cycles: 1

Encoding: | 0000 | 1rrr |

Operation: INC
(Rn) ← (Rn) + 1

INC direct

Bytes: 2

Cycles: 1

Encoding: | 0000 | 0101 | | direct address |

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

INC @Ri

Bytes: 1

Cycles: 1

Encoding: | 0000 | 011i |

Operation: INC
((Ri)) ← ((Ri)) + 1

INC DPTR

Function: Increment Data Pointer

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

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

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

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

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

Bytes: 1

Cycles: 2

Encoding: | 1010 | 0011 |

Operation: INC
(DPTR) ← (DPTR) + 1

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

JB bit,rel

Function: Jump if Bit set

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

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

```
JB   P1.2,LABEL1

JB   ACC.2,LABEL2
```

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

Bytes: 3

Cycles: 2

Encoding:

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

Operation:

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

JBC bit,rel

Function: Jump if Bit is set and Clear bit

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

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

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

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

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Bytes: 3

Cycles: 2

Encoding: | 0 0 0 1 | 0 0 0 0 | bit address | rel. address |

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

JC rel

Function: Jump if Carry is set

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

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

```
JC    LABEL1
CPL   C
JC    LABEL 2
```

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

Bytes: 2

Cycles: 2

Encoding: | 0 1 0 0 | 0 0 0 0 | rel. address |

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

JMP @A + DPTR

Function: Jump indirect

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

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

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

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

Bytes: 1

Cycles: 2

Encoding: | 0 1 1 1 | 0 0 1 1 |

Operation: JMP
(PC) ← (A) + (DPTR)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

JNB bit,rel

Function: Jump if Bit Not set

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

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

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

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

Bytes: 3

Cycles: 2

Encoding: | 0 0 1 1 | 0 0 0 0 | bit address | rel. address |

Operation:

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

JNC rel

Function: Jump if Carry not set

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

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

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

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

Bytes: 2

Cycles: 2

Encoding: | 0 1 0 1 | 0 0 0 0 | rel. address |

Operation:

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


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

JNZ rel

Function: Jump if Accumulator Not Zero

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

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

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

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

Bytes: 2

Cycles: 2

Encoding: | 0 1 1 1 | 0 0 0 0 | | rel. address |

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

JZ rel

Function: Jump if Accumulator Zero

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

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

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

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

Bytes: 2

Cycles: 2

Encoding: | 0 1 1 0 | 0 0 0 0 | | rel. address |

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

LCALL addr16

Function: Long call

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

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

LCALL SUBRTN

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

Bytes: 3

Cycles: 2

Encoding:

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

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

LJMP addr16

Function: Long Jump

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

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

LJMP JMPADR

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

Bytes: 3

Cycles: 2

Encoding:

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

Operation: LJMP
(PC) ← $addr_{15-0}$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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

Function: Move byte variable

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

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

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

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

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

MOV A,Rn

Bytes: 1

Cycles: 1

Encoding:

1 1 1 0	1 r r r

Operation: MOV
(A) ← (Rn)

***MOV A,direct**

Bytes: 2

Cycles: 1

Encoding:

1 1 1 0	0 1 0 1	direct address

Operation: MOV
(A) ← (direct)

***MOV A,ACC is not a valid instruction.**

MCS-51

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

MOV A,@Ri

Bytes: 1

Cycles: 1

Encoding: | 1 1 1 0 | 0 1 1 i |

Operation: MOV
(A) ← ((Ri))

MOV A,#data

Bytes: 2

Cycles: 1

Encoding: | 0 1 1 1 | 0 1 0 0 | | immediate data |

Operation: MOV
(A) ← #data

MOV Rn,A

Bytes: 1

Cycles: 1

Encoding: | 1 1 1 1 | 1 r r r |

Operation: MOV
(Rn) ← (A)

MOV Rn,direct

Bytes: 2

Cycles: 2

Encoding: | 1 0 1 0 | 1 r r r | | direct addr. |

Operation: MOV
(Rn) ← (direct)

MOV Rn,#data

Bytes: 2

Cycles: 1

Encoding: | 0 1 1 1 | 1 r r r | | immediate data |

Operation: MOV
(Rn) ← #data

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

MOV direct,A

Bytes: 2

Cycles: 1

Encoding: | 1 1 1 1 | 0 1 0 1 | | direct address |

Operation: MOV
(direct) ← (A)

MOV direct,Rn

Bytes: 2

Cycles: 2

Encoding: | 1 0 0 0 | 1 r r r | | direct address |

Operation: MOV
(direct) ← (Rn)

MOV direct,direct

Bytes: 3

Cycles: 2

Encoding: | 1 0 0 0 | 0 1 0 1 | | dir. addr. (src) | | dir. addr. (dest) |

Operation: MOV
(direct) ← (direct)

MOV direct,@Ri

Bytes: 2

Cycles: 2

Encoding: | 1 0 0 0 | 0 1 1 i | | direct addr. |

Operation: MOV
(direct) ← ((Ri))

MOV direct,#data

Bytes: 3

Cycles: 2

Encoding: | 0 1 1 1 | 0 1 0 1 | | direct address | | immediate data |

Operation: MOV
(direct) ← #data

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

MOV @Ri,A

Bytes: 1

Cycles: 1

Encoding: | 1 1 1 1 | 0 1 1 i |

Operation: MOV
((Ri)) ← (A)

MOV @Ri,direct

Bytes: 2

Cycles: 2

Encoding: | 1 0 1 0 | 0 1 1 i | | direct addr. |

Operation: MOV
((Ri)) ← (direct)

MOV @Ri,#data

Bytes: 2

Cycles: 1

Encoding: | 0 1 1 1 | 0 1 1 i | | immediate data |

Operation: MOV
((RI)) ← #data

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

Function: Move bit data

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

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

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

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

MOV C,bit

Bytes: 2

Cycles: 1

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

Operation: MOV
(C) ← (bit)

MOV bit,C

Bytes: 2

Cycles: 2

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

Operation: MOV
(bit) ← (C)

MOV DPTR,#data16

Function: Load Data Pointer with a 16-bit constant

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

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

Example: The instruction,

MOV DPTR,#1234H

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

Bytes: 3

Cycles: 2

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

Operation: MOV
(DPTR) ← $\#data_{15\text{-}0}$
DPH □ DPL ← $\#data_{15\text{-}8}$ □ $\#data_{7\text{-}0}$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

MOVC A,@A+ <base-reg>

Function: Move Code byte

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

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

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

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

MOVC A,@A+DPTR

Bytes: 1

Cycles: 2

Encoding: | 1 0 0 1 | 0 0 1 1 |

Operation: MOVC
(A) ← ((A) + (DPTR))

MOVC A,@A + PC

Bytes: 1

Cycles: 2

Encoding: | 1 0 0 0 | 0 0 1 1 |

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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

Function: Move External

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

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

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

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

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

```
MOVX   A,@R1

MOVX   @R0,A
```

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

MOVX A,@Ri

Bytes: 1

Cycles: 2

Encoding:

1 1 1 0	0 0 1 i

Operation: MOVX
(A) ← ((Ri))

MOVX A,@DPTR

Bytes: 1

Cycles: 2

Encoding:

1 1 1 0	0 0 0 0

Operation: MOVX
(A) ← ((DPTR))

MOVX @Ri,A

Bytes: 1

Cycles: 2

Encoding:

1 1 1 1	0 0 1 i

Operation: MOVX
((Ri)) ← (A)

MOVX @DPTR,A

Bytes: 1

Cycles: 2

Encoding:

1 1 1 1	0 0 0 0

Operation: MOVX
(DPTR) ← (A)

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

MUL AB

Function: Multiply

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

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

```
MUL   AB
```

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

Bytes: 1

Cycles: 4

Encoding: | 1 0 1 0 | 0 1 0 0 |

Operation: MUL
$(A)_{7-0} \leftarrow (A) \text{ X } (B)$
$(B)_{15-8}$

NOP

Function: No Operation

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

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

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

Bytes: 1

Cycles: 1

Encoding: | 0 0 0 0 | 0 0 0 0 |

Operation: NOP
$(PC) \leftarrow (PC) + 1$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ORL <dest-byte> <src-byte>

Function: Logical-OR for byte variables

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

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

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

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

ORL A,R0

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

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

ORL P1,#00110010B

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

ORL A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 1 0 0	1 r r r

Operation: ORL
(A) ← (A) ∨ (Rn)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ORL A,direct

Bytes: 2

Cycles: 1

Encoding: | 0 1 0 0 | 0 1 0 1 | direct address |

Operation: ORL
(A) ← (A) V (direct)

ORL A,@Ri

Bytes: 1

Cycles: 1

Encoding: | 0 1 0 0 | 0 1 1 i |

Operation: ORL
(A) ← (A) V ((Ri))

ORL A,#data

Bytes: 2

Cycles: 1

Encoding: | 0 1 0 0 | 0 1 0 0 | immediate data |

Operation: ORL
(A) ← (A) V #data

ORL direct,A

Bytes: 2

Cycles: 1

Encoding: | 0 1 0 0 | 0 0 1 0 | direct address |

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

ORL direct,#data

Bytes: 3

Cycles: 2

Encoding: | 0 1 0 0 | 0 0 1 1 | direct addr. | immediate data |

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ORL C,<src-bit>

Function: Logical-OR for bit variables

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

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

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

ORL C,bit

Bytes: 2

Cycles: 2

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

Operation: ORL
(C) ← (C) V (bit)

ORL C,/bit

Bytes: 2

Cycles: 2

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

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

MCS-51

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

POP direct

Function: Pop from stack.

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

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

```
POP   DPH
POP   DPL
```

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

```
POP   SP
```

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

Bytes: 2

Cycles: 2

Encoding: | 1 1 0 1 | 0 0 0 0 | | direct address |

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

PUSH direct

Function: Push onto stack

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

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

```
PUSH   DPL
PUSH   DPH
```

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

Bytes: 2

Cycles: 2

Encoding: | 1 1 0 0 | 0 0 0 0 | | direct address |

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

RET

Function: Return from subroutine

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

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

RET

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

Bytes: 1

Cycles: 2

Encoding:

0 0 1 0	0 0 1 0

Operation: RET
$(PC_{15-8}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$
$(PC_{7-0}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$

RETI

Function: Return from interrupt

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

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

RETI

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

Bytes: 1

Cycles: 2

Encoding:

0 0 1 1	0 0 1 0

Operation: RETI
$(PC_{15-8}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$
$(PC_{7-0}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

RL A

Function: Rotate Accumulator Left

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

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

RL A

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

Bytes: 1

Cycles: 1

Encoding:

0 0 1 0	0 0 1 1

Operation: RL
$(A_n + 1) \leftarrow (An) \quad n = 0 - 6$
$(A0) \leftarrow (A7)$

RLC A

Function: Rotate Accumulator Left through the Carry flag

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

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

RLC A

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

Bytes: 1

Cycles: 1

Encoding:

0 0 1 1	0 0 1 1

Operation: RLC
$(An + 1) \leftarrow (An) \quad n = 0 - 6$
$(A0) \leftarrow (C)$
$(C) \leftarrow (A7)$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

RR A

Function: Rotate Accumulator Right

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

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

RR A

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

Bytes: 1

Cycles: 1

Encoding: | 0 0 0 0 | 0 0 1 1 |

Operation: RR
$(An) \leftarrow (A_n + 1) \quad n = 0 - 6$
$(A7) \leftarrow (A0)$

RRC A

Function: Rotate Accumulator Right through Carry flag

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

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

RRC A

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

Bytes: 1

Cycles: 1

Encoding: | 0 0 0 1 | 0 0 1 1 |

Operation: RRC
$(An) \leftarrow (An + 1) \quad n = 0 - 6$
$(A7) \leftarrow (C)$
$(C) \leftarrow (A0)$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

SETB <bit>

Function: Set Bit

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

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

SETB C

SETB P1.0

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

SETB C

Bytes: 1

Cycles: 1

Encoding:

1 1 0 1	0 0 1 1

Operation: SETB
(C) ← 1

SETB bit

Bytes: 2

Cycles: 1

Encoding:

1 1 0 1	0 0 1 0		bit address

Operation: SETB
(bit) ← 1

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

SJMP rel

Function: Short Jump

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

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

SJMP RELADR

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

(*Note:* Under the above conditions the instruction following SJMP will be at 102H. Therefore, the displacement byte of the instruction will be the relative offset (0123H-0102H) = 21H. Put another way, an SJMP with a displacement of 0FEH would be a one-instruction infinite loop.)

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	0 0 0 0		rel. address

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

SUBB A,<src-byte>

Function: Subtract with borrow

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

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

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

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

SUBB A,R2

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

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

SUBB A,Rn

Bytes: 1

Cycles: 1

Encoding:

1 0 0 1	1 r r r

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

SUBB A,direct

Bytes: 2

Cycles: 1

Encoding: | 1 0 0 1 | 0 1 0 1 | | direct address |

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

SUBB A,@Ri

Bytes: 1

Cycles: 1

Encoding: | 1 0 0 1 | 0 1 1 i |

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

SUBB A,#data

Bytes: 2

Cycles: 1

Encoding: | 1 0 0 1 | 0 1 0 0 | | immediate data |

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

SWAP A

Function: Swap nibbles within the Accumulator

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

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

SWAP A

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

Bytes: 1

Cycles: 1

Encoding: | 1 1 0 0 | 0 1 0 0 |

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

XCH A,<byte>

Function: Exchange Accumulator with byte variable

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

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

XCH A,@R0

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

XCH A,Rn

Bytes: 1

Cycles: 1

Encoding: | 1 1 0 0 | 1 r r r |

Operation: XCH
(A) ⇄ (Rn)

XCH A,direct

Bytes: 2

Cycles: 1

Encoding: | 1 1 0 0 | 0 1 0 1 | | direct address |

Operation: XCH
(A) ⇄ (direct)

XCH A,@Ri

Bytes: 1

Cycles: 1

Encoding: | 1 1 0 0 | 0 1 1 i |

Operation: XCH
(A) ⇄ ((Ri))

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

XRL A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 1 1 0	1 r r r

Operation: XRL
$(A) \leftarrow (A) \veebar (Rn)$

XRL A,direct

Bytes: 2

Cycles: 1

Encoding:

0 1 1 0	0 1 0 1	direct address

Operation: XRL
$(A) \leftarrow (A) \veebar (direct)$

XRL A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 1 1 0	0 1 1 i

Operation: XRL
$(A) \leftarrow (A) \veebar ((Ri))$

XRL A,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 1 0	0 1 0 0	immediate data

Operation: XRL
$(A) \leftarrow (A) \veebar \#data$

XRL direct,A

Bytes: 2

Cycles: 1

Encoding:

0 1 1 0	0 0 1 0	direct address

Operation: XRL
$(direct) \leftarrow (direct) \veebar (A)$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

XRL direct, # data

Bytes: 3

Cycles: 2

Encoding: | 0 1 1 0 | 0 0 1 1 | | direct address | | immediate data |

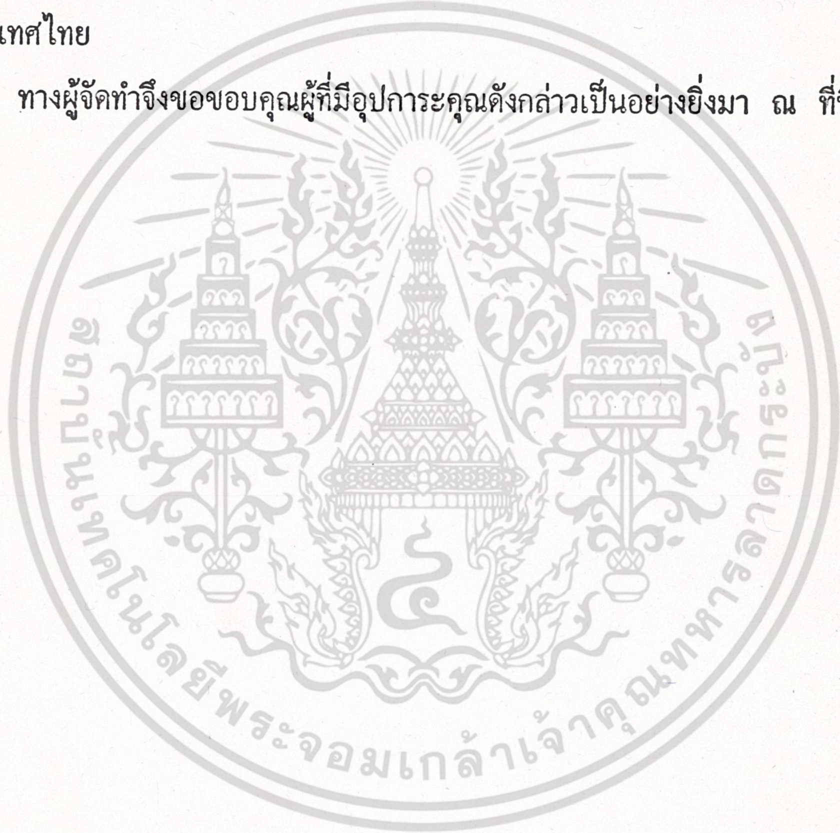
Operation: XRL
(direct) ← (direct) ⊻ # data

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

กิตติประกาศ

ปริญญานิพนธ์ฉบับนี้ได้จัดทำขึ้นและสำเร็จลุล่วงสมบูรณ์ได้โดยดีโดยได้รับความช่วยเหลือและอนุเคราะห์ทางด้านข้อมูลและเอกสารต่าง ๆ จากบริษัทศิลารีเสิร์ซ จำกัด โดยเฉพาะ คุณไพฑูรย์ Solf ware Engineer ของบริษัท และทางเจ้าที่ที่ทำการโทรสาร ของการสื่อสารแห่งประเทศไทย นี้ให้ใช้ห้อง Lab เป็นห้องทดลองจนสำเร็จ ส่วนทางด้านการทำรายงานได้รับความช่วยเหลือเป็นอย่างดียิ่งจากเจ้าหน้าที่ศูนย์ส่งเสริมพระพุทธศาสนาแห่งประเทศไทย

ทางผู้จัดทำจึงขอขอบคุณผู้ที่มีอุปการะคุณดังกล่าวเป็นอย่างยิ่งมา ณ ที่นี้ด้วย



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หนังสืออ้างอิง

๑. ผศ. พิพัฒน์ เลาหสงคราม "ไมโครคอนโทรลเลอร์ Mcs-51 วิศวสาร ลาดกระบัง หน้า ๔๙-๖๐ , ฉบับที่ ๕ ปีที่ ๙ ๒๕๓๒

๒. สมยศ ภูงามแปลง "Mcs-51 ไมโครคอนโทรลเลอร์ แบบชิพเดี่ยว" เซมิคอนดัคเตอร์อิเล็กทรอนิคส์ หน้า ๒๖๔ - ๒๖๙ , ฉบับที่ ๙๓,๒๕๓๒

๓. " INTEL CO,LTD." คู่มือไอซี ไมโครโปเซสเซอร์ Mcs-51"

๔. INTEL CO,LTD" Microprocessor data book Mcs-51", 1989

๕. RODNAY ZAKS AND AUSTIN LESEA "MICROPROCESSOR INTERFACING TECHNIOUES"

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้