Programmable Power Supply

โดย

นายสถิตย์พร นวเศรษฐกุล

นายเอนก สืบปันอุด

ปริญญานิพนธ์วิศวกรรมศาสตร์บัณฑิต

ภาควิชาเทคนิคอุตสาหกรรม คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2532

ปีการศึกษา 2532

หัวข้อปริญญานิพนธ์ PROGRAMMABLE POWER SUPPLY

โดย นาย สถิตย์พร นวเศรษฐกุล 323419
นาย เอนก สืบปันอุด 323437

ภาควิชา..... เทคนิคอุตสาหกรรม

อาจารย์ที่ปรึกษา..... ผศ. นิกร สุขุตมตันติ

ภาควิชาเทคนิคอุตสาหกรรม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยี่ พระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง อนุมัติให้นับปริญญานิพนธ์นี้เป็นส่วนหนึ่ง ของวิชาการศึกษาปริญญาบัณฑิต

..... หัวหน้าภาควิชาเทคนิคอุตสาหกรรม
(.....)

คณะกรรมการ

.....ประธานกรรมการ
(.....)
.....กรรมการ
(.....)
.....กรรมการ
(.....)

PROGRAMMABLE POWER SUPPLYY

นาย สลิตย์พร นวเศรษฐกุล
นาย เอนก สืบปันอุด

อาจารย์ที่ปรึกษา

ผศ. นิกร สุขุตมตันติ
ปีการศึกษา 2532

บทคัดย่อ

ในปัจจุบันนี้โลกอิเล็กทรอนิคส์ก้าวหน้าไปมาก อุปกรณ์และเครื่องมือที่ใช้อยู่เกือบทุกชนิดต้องมีคอมพิวเตอร์คอยเป็นตัวควบคุม ผู้จัดทำเห็นว่าถ้านำแหล่งจ่ายไฟมาดัดแปลงโดยใช้ ซี พี ยู มาควบคุมโดยใช้การกดปุ่มคีย์ แล้วให้แรงไฟตามปุ่มที่กดจะดีกว่าการปรับไฟโดยใช้มือปรับหมุน ฉะนั้นผู้จัดทำจึงนำความคิดนี้มาจัดทำขึ้นมา

ผู้จัดทำ

วัตถุประสงค์

1. เพื่อประยุกต์การใช้งานของไมโครโปรเซสเซอร์ในการควบคุมต่างๆ

2. เพื่อศึกษาการทำและปัญหาต่างๆ ที่จะเกิดขึ้นขณะที่ใช้งานจริง

3. เพื่อทดสอบทฤษฎีว่าระหว่างใช้งานจริงจะมีปัญหาอะไรบ้าง

สำนัก

ในปัจจุบันการดำรงชีพของมนุษย์เกี่ยวข้องกับเครื่องมือ อุปกรณ์ทางอิเล็คทรอนิคส์เป็นอย่างมาก ซึ่งคุณสมบัติของเครื่องมือทางอิเล็คทรอนิคส์นั้นขึ้นอยู่กับปัจจัยหลายอย่างด้วยกัน โดยเฉพาะแหล่งจ่ายไฟ ซึ่งแหล่งจ่ายไฟที่ดีนั้นต้องมีแรงดันไฟที่สม่ำเสมอไม่มีสัญญาณรบกวน โดยเฉพาะแหล่งจ่ายพลังงานที่ต้องจ่ายให้ IC TTL

ชุดควบคุมแหล่งจ่ายไฟด้วยไมโครโปรเซสเซอร์ (programmable power supply) ได้จัดทำขึ้นเพื่อให้เป็นชุดจ่ายไฟแรงดันไฟฟ้าที่มีคุณภาพแบบหนึ่ง ซึ่งจะทำให้สะดวกในการใช้งาน ขั้นตอนการประดิษฐ์ ตลอดจนการออกแบบอาศัยหลักการวงจรดิจิตอล วรรจรอนาล็อก ให้สามารถทำงานได้ โดยมีไมโครโปรเซสเซอร์เป็นตัวควบคุมการทำงาน

ชุดแหล่งจ่ายไฟนี้ สามารถปรับแรงดันได้ตั้งแต่ 0-15 Volts กระแส 1.5 AMP การใช้งานเพียงแต่ป้อนแรงดันที่ต้องการ ผ่านคีย์บอร์ด DAC ออกแรงดันไฟ และวงจรป้องกันในกรณีเกิดกระแสเกินวงจร (overload protection)

สารบัญ

บทที่ 1

การเขียนโปรแกรมสำหรับไมโครคอมพิวเตอร์

การทำงานของเครื่องคอมพิวเตอร์เหล่านี้ไม่ว่าจะเป็นเครื่องคอมพิวเตอร์จะต้องทำงานภายใต้เงื่อนไขของโปรแกรมที่สั่งงาน และถ้าพิจารณาให้ลึกซึ้งลงไปจะเห็นว่าระดับที่เครื่องจะรับรู้และกระทำตามคำสั่งได้เป็นคำสั่งในลักษณะภาษาเครื่อง (Machine Instruction) ที่เขียนเป็นเลขไบนารี่ ซึ่งจะรับคำสั่งเหล่านี้แล้วตีความในหน่วยควบคุมเพื่อกระทำงานตามคำสั่งนี้ เครื่องคอมพิวเตอร์ทั่วๆไปก็มีคำสั่งภาษาเครื่องที่แตกต่างกันขึ้นอยู่กับผู้ออกแบบโครงสร้าง

เรามาลองรู้จักว่าในการที่เราป้อนโปรแกรมด้วยระดับสูง เช่น FORTRAN หรือ BASIC อาศัยตัวแปรโปรแกรมเหล่านี้จนเป็นภาษาเครื่องแล้วจึงจะทำงานได้

ในทำนองเดียวกันการจัดการทำงานของเครื่องต่างๆ เช่นจัดให้ภาคแสดงผลลัพท์ตามการคำนวณได้ หรือรับสั่งบางอย่างจากคีย์บอร์ด การรับรู้วิธีการควบคุมบางอย่างเครื่องก็จะต้องรับรู้ภายใต้โปรแกรมแปลโปรแกรมเหล่านี้จนเป็นภาษาเครื่องแล้วจึงจะทำงานได้ แต่ถ้าเราให้โปรแกรมเครื่องรับรู้ให้ทำงานได้ในส่วนจำกัดเครื่องก็จะทำงานได้ในขอบเขตจำกัด เช่น โปรแกรมมอนิเตอร์สำหรับเครื่องไมโครคอมพิวเตอร์แผงเดียว

<u>ลักษณะของโปรแกรม</u>

จากที่เราได้ศึกษามาแล้วในบทก่อนเกี่ยวกับชุดคำสั่งของ 8080 โดยแต่ละคำสั่งจะมีรหัสที่แทนหรือเป็นรหัสเครื่องรับรู้ในการสั่งงาน การเขียนโปรแกรมในลักษณะนี้เรียกภาษาเครื่อง หรือออปเจคโปรแกรม

```
00111010
01100000
00000000
01000111
00111010
01100001
00000000
10000000
00110010
01100010
00000000
```

ออปเจคโปรแกรม

การเขียนโปรแกรมด้วยภาษาเครื่องเหมาะสำหรับเครื่องไมโครคอมพิวเตอร์ที่ผู้ออกแบบลดข้อยุ่งยากนัก แต่ถ้าเราต้องการเขียนโปรแกรมที่ยาว ๆและยุ่งยากซับซ้อน การเขียนด้วยเลขฐานไบนารี่ก็มีปัญหาเช่น

1. เป็นโปรแกรมที่ยากต่อการเขียน การแก้ไขและตรวจสอบโปรแกรมเพราะเป็นเลข 0 และ 1 หมดทำให้เราดูยาก

2. การป้อนโปรแกรมจะเป็นไปด้วยความลำบากทำให้เสียเวลาในการป้อนโปรแกรม

3. การอ่านโปรแกรมเป็นไปด้วยความลำบากทำให้เสียโอกาสในการเข้าใจตัวโปรแกรมยากด้วย

4. ต้องใช้เวลาในการเขียนโปรแกรมนาน

5. ผู้เขียนโปรแกรมจะต้องระมัดระวังในการเขียนเป็นพิเศษเพราะโอกาสผิดพลาดมีได้ง่ายมาก และเมื่อผิดแล้วจะหาทางแก้ไขได้ยาก

จากเหตุผลดังกล่าวจึงได้มีผู้ออกแบบตัวแปรรหัสโดยการป้อนเป็นรหัสที่เราเข้าใจได้ง่ายแล้วให้เครื่องแปลเป็นภาษาเครื่องอีกต่อหนึ่ง แนวทางการรวมเลขไบนารี่เข้าเป็นกลุ่มให้จดจำได้ง่าย ลักษณะการเขียนโปรแกรมสามารถเขียนได้ดังนี้

```
3A
60
00
47
3A
00
80
32
62
00
```

โปรแกรมดังกล่าวเมื่อเขียนด้วยตัวเลขฐานสิบหกการป้อนโปรแกรมเป็นไปได้รวดเร็วยิ่งขึ้นและสามารถตรวจสอบโปรแกรมได้ดีกว่าโดยการเปลี่ยนจากเลขฐานสิบหกที่เรากดแป้นจะเกิดขึ้นในวงจรระบบอาร์ดแวร์ หรือซอร์แวร์ ที่ผู้ออกแบบโครงไว้

แต่อย่างไรก็ตามถ้าหากว่ามีใครก็ตามเอาตัวโปรแกรมที่เขียนด้วยเลขฐานสิบหกมาให้เราติความว่าในแต่ละคำสั่งหมายถึงอะไร เพราะรหัสเหล่านี้ยากต่อการจดจำหรือติความหมาย

จึงมีผู้พยายามที่จะหาความสะดวกสบายให้กับผู้เขียนในโปรแกรมด้วยการกำหนดตัวอักษรที่เหมาะสมกับคำสั่งที่เรียกว่า นีโมนิค ลักษณะของนีโมนิคจะกำหนดกันเป็นมาตรฐานสำหรับไมโครโปรเซสเซอร์แต่ละเบอร์ แต่มักจะยึดถือหลักการให้เข้าใจว่าง่าย ตัวอย่างโปรแกรมที่มาแล้วสามารถเขียนได้ดังนี้

```
      LDA
       60
       00
     MOV B,A
      LDA
       61
       00
     ADD B
      STA
       62
       00
```

การเขียนโปรแกรมในรูปของตัวอักษรที่เราเข้าใจกันง่ายทำให้สามารถตรวจแก้ไขหรือเข้าใจในการทำงานของโปรแกรมได้ชัดเจน

ภาษาแอสเซมบลี้ (Assembly Language)

การวางรูปโปรแกรมดังกล่าวยังต้องเกี่ยวข้องโดยตรงกับระดับภาษาเครื่องแต่เราไม่ต้องยุ่งเกี่ยวกับตัวเลขไบนารี่หรือตำแหน่งต่างๆ ของแอดเดรส มากนัก ลักษณะของรูปแบบโปรแกรมจะเป็นดังนี้

```
เลเบล      นีโมนิค     โอเปอร์แรนด์          หมายเหตุ
START      LDA         VAL 1          :LOAD FIRST NUMBER INTO A
           MOV         B,A            :SAVE IN B
           LDA         VAL 2          :LOAD SECOND NUMBER INTO A
           ADD         B              :ADD FIRST NUMBER TO A
```

เลเบล	นีโมนิค	โอเปอร์แรนด์	หมายเหตุ
	STA	SUM	:STORE SUM
NEXT	?	?	:NEXT INSTUCTION
VALI	DS		
VALI	DS		
SUM	DS		

ลักษณะของภาษาแซมบลี้จุประกอบด้วยฟิลด์ที่สำคัญ ดังนี้

<u>เลเบล</u> เป็นฟิลด์ที่มีไว้สำหรับอ้างถึงในส่วนตัวโปรมแกรม เลเบลเปรียบเสมือนกับค่าตำแหน่งแอดเดรสที่จะอ้างถึง ส่วนของเลเบลจะเขียนด้วยลักษณะของอักษรภาษาอังกฤษและจะจัดวางไว้ในฟิลด์แรก

การที่เรากำหนดเลเบลเป็นตัวอักษรมีข้อดีทำให้ในขณะที่เรากำลังเขียนโปรแกรม เราไม่ต้องพะวงถึงตำแหน่งของแอดเดรสจริง ๆ ในหน่วยความจำที่จะเก็บโปรแกรมไว้ เพราะสิ่งเหล่านี้จะได้รับการกำหนดโดยขบวนการแอสเซมเบลอร์

<u>นีโมนิคฟิลด์</u> เป็นฟิวล์ที่ใช้แสดงการกระทำของคำสั่ง ลักษณะของนีโนนิคจะเป็นคำเฉพาะที่ที่ได้รับการกำหนดขึ้นมาและรับรู้กันโดยทั่วไปของ 8080 เรากำหนดให้ ADD หมายถึงการบวก CMC หมายถึงการคอมพลีเมนท์ค่าในรีจิสเตอร์ A

<u>โอเปอร์แรนด์ฟิลด์</u> เป็นฟิลด์ที่ใช้เก็บค่าตัวแปรหรือข้อมูลที่จะนำมากระทำตามคำสั่งที่อยู่ในนีโมนิคฟิลด์ เช่น

START LDA VAL 1 ; ตัว VAL 1 หมายถึงโอเปอร์แรนด์ที่จะได้รับการโหลดมาเก็บไว้ในรีจิสเตอร์ คอมเมนต์ (comment) หรือฟิลด์หมายเหตุ เป็นฟิลด์ที่ไว้สำหรับเขียนอธิบายลักษณะของคำสั่งหรือโปรแกรมว่าส่วนนี้กำลังทำอะไรอยู่เพื่อให้การอ่านโปรแกรมเป็นไปได้ง่าย

<u>ตัวอย่างโปรแกรมอย่างง่าย</u>

เพื่อให้เข้าใจในการเขียนโปรแกรมของ 8080 ดียิ่งขึ้นคราวนี้ลองมาพิจารณาตัวอย่างการเขียนโปรแกรมดังต่อไปนี้

ตัวอย่าง เป็นการกระทำการคอมพลีเมนท์ข้อมูลที่เก็บไว้ในตำแหน่งหน่วยความจำที่ 4016 แล้วเก็บผลลัพท์ที่ได้ในตำแหน่งหน่วยความจำที่ 4116

ภาษาเครื่องเลขฐานสิบหก		ภาษาแอสเซมเบลอร์			คอมเมนท์
ตำแหน่งหน่วยความจำ	ข้อมูลในหน่วยความจำ	เลเบล	นิโมนิค	โอเปอร์แรนด์	
00	3A	START	LDA	40H	:โหลดข้อมูลจากตำแหน่ง ๔๐ เข้าเก็บรีจิสเตอร์ A
02	00				
03	2F		CMA		:คอมพลีเมนท์ข้อมูล A
04	32		STA	41H	:เก็บผลลัพท์ไว้ที่ตำแหน่ง 4116
05	41				
06	00				
07	C3	HEAR	JMP	HEAR	:ทำซ้ำคำสั่งไปเรื่อยๆ
08	07				
09	00				

ตัวอย่าง จงเขียนโปรแกรมแสดงการบวกข้อมูลในตำแหน่ง หน่วยความจำ 4016แล้วเก็บผลลัพท์ไว้ในตำแหน่งที่ 4216

แอดเดรส	ข้อมูล	เลเบล	นิโมนิค	โอเปอร์เตอร์	คอมเมนท์
00	3A		LDA	40H	:GET DATA
01	40				
02	00				
03	87		ADD	A	:SHIFT DATA LEFT
04	32		STA	41H	:STORE RESULT
05	41				
06	00				
07	C3	HEAR	JMP	HEAR	
08	07				
09	00				

ตัวอย่าง จงเขียนโปรแกรมทำการเลื่อนข้อมูลที่อยู่แอดเดรส 4016 ไปซ้าย 1 บิทแล้วแล้วเก็บผลลัพท์ไว้

แอดเดรส	ข้อมูล	เลเบล	นิโมนิค	โอเปอร์แรนด์	คอมเมนท์
00	3A				
01	40				
02	00				
03	87		ADD	A	
04	32		STA		
05	41				
06	00				
07	C3	HEAR	JMP	HEAR	
08	07				
09	00				

จากผลของการกระทำตามโปรแกรมข้างบน ถ้าสมมุติว่าเดิมหน่วยความจำตำแหน่ง (40)16 เก็บข้อมูลด้วยตัวเลข 6F ผลลัพท์ที่จะได้จะปรากฏที่ตำแหน่ง (41)16 ด้วยค่า DE ลักษณะการเลื่อนบิทในลักษณะนี้จะมีความหมายในการคูณด้วย 2 เราจึงใช้วิธีการบวกตัวของมันเองหนึ่งครั้งได้

ตัวอย่าง จงแสดงการเขียนโปแกรมเคลียร์ข้อมูล ในหน่วยความจำตำแหน่งที่ 4016

แอดเดรส	ข้อมูล	เลเบล	นิโมนิค	โอเปอร์แรนด์	คอมเมนท์
00	97		SUB	A	;
01	32		STA	40H	; CLEAR LOCAATION 40
02	40				
03	00				
04	C3	HERE	JMP	HERE	
05	04				
06	00				

ตัวอย่าง ในแบ่งข้อมูลของหน่วยความจำตำแหน่งที่ 40 ออกเป็นสองส่วน ๆ ละสี่สิบบิทแล้วเก็บไวในหน่วยความจำตำแหน่งที่ 41 และ 42 โดยให้ส่วน 4 บิท มีนัยสำคัญมากที่สุดจากตำแหน่ง 40 มาเก็บไว้ที่ตำแหน่งที่มีความสำคัญน้อยที่สุดของตำแหน่ง 41 และส่วนที่ มีนัยสำคัยน้อยที่สุด 4 บิท ของตำแหน่ง 40 มาเก็บไว้ที่ตำแหน่งที่มีนัยสำคัญน้อยสุดของตำแหน่ง 42 ให้ทำการเคลียร์สี่บิทที่มีนัยสำคัญมากสุดของตำแหน่ง 4116 และ 4216

แอดเดรส	ข้อมูล	เลเบล	นีโมนิค	โอเปอร์แรนด์	คอมเมนท์
00	21	START	LXI	H,40H	;GET DATA
01	40				
02	00				
03	7E		MOV	A,M	;GET DATA
04	47		MOV	B,A	
05	0F		RRC		;SHIFT DATA RIGHT 4 TIMES
06	0F		RRC		
07	0F		RRC		
08	0F		RRC		
09	E6		ANI	00001111B	;MASK OFF LSBS
0A	0F				
0B	23		INX	H	
0C	77		MOV	M,A	;STORE MSBS
0D	78		MOV	A,B	;RESTORE ORIGINAL DATA
0E	E6		ANI	00001111B	;MASK OFF MSBS
0F	0F				
10	23		INX	H	
11	77		MOV	M,A	;STORE LSBS
12	C3	HEAR	JMP	HEAR	;STORE LSBS
13	12				
14	00				

ตัวอย่าง สมมติว่า2 ตัว จงเขียนโปรแกรมการตรวจสอบข้อมูลสองตัวนี้ว่าตัวใดมีค่ามากกว่า

ในที่นี้เราสมมติว่าข้อมูลที่ต้องการตรวจสอบเก็บไว้ที่หน่วยความจำตำแหน่งแอดเดรส 40 16 และ 4116 ตัวโปรแกรมที่เขียนได้จะมีลักษณะดังนี้

แอดเดรส	ข้อมูล	เลบิล	นีโมนิค	โอเปอร์แรนด์	คอมเมนท์
00	21		LXI	H,40H	;GET FIRST OPERAND
01	40				
02	00		MOV	A,M	;GET FIRST OPERAND
03	7E		INX	H	
04	23		CMP	M	;IS SECOND OPERAND LARGER
05	BE				
06	D2				
07	0A				
08	00				
09	7E		MOV	A,M	;YES,GET SECOND INSTEAD
0A	23	DONE	INX	H	
0B	77		MOV	M,A	;STORE LARGER OPERAND
0C	C3				
0D	0C	HEAR	JMP	HEAR	
0E	00				

จากโปรแกรมนี้อาศัยหลักการทดสอบแฟลกที่มีผลจากการเปรียบเทียบในคำสั่ง CMP โดยการนำข้อมูลจากหน่วยความจำเปรียบเทียบกับข้อมูลในรีจิสเตอร์ A

ตัวอย่าง สมมติว่า มีตัวเลขขนาด 16 บิท สองตัว ตัวที่หนึ่งเก็บไว้ที่หน่วยความจำตำแหน่งที่ 4016 และ 4116 ส่วนตัวที่สอง เก็บไว้ที่ตำแหน่ง 4216 และ 4316 จงเขียนโปรแกรมบวกข้อมูล 16 บิท นี้แล้วนำผลลัพท์ที่บวกได้เก็บไว้ในหน่วยความจำตำแหน่ง 4416 และ 4516

แอดเดรส	ข้อมูล	เลเบล	นีโมนิค	โอเปอร์แรนด์	คอมเมนท์
00	2A		LHLD	40H	;LOAD TWO BYTE INTO HL REG
01	40				
02	00				
03	EB		XCHG		;EXCHANGE HL WITH DE
04	2A		LHLD	42H	;LOAD TWO BYTE INTO HL REG
05	42				
06	00				

```
08        22          SHLD     44H     ;STORE RESULT INTO MEN
09        44
0A        00
0B        C3   HEAR   JMP      HERE
0C        0B
0D        00
```

ตัวอย่างาข้างบนถ้าสมมติว่าแต่เดิมข้อมูลในหน่วยความจำต่างๆ มีดังนี้

(40) =2A

(41) =67

(42) =F8

(43) =14

ผลลัพท์ที่ได้จากโปรแกรมคือ

(44) =22

(45) =7C

<u>การสร้างตารางและการมองหาตารางเบื้องต้น</u>

ตัวอย่าง สมมติว่าต้องการให้เครื่องคำนวณหาค่ากำลังสองของตัวเลขจาก 0-7 ในการเขียนโปรแกรมเราอาจทำได้ด้วยการสร้างตารางขึ้นที่ส่วนหนึ่งส่วนใดของหน่วยความจำ เพื่อเก็บผลลัพท์ไว้ก่อน และเมื่อต้องการหาค่ากำลังสองก็ใช้วิธีมองหาตารางโดยอาศัยวิธีการสร้างอินแดด เพื่อหาว่าผลลัพท์คืออะไร เมื่อเป็นเช่นนี้การกำหนดค่าในตารางเราต้องรู้ค่าที่สัมพันธ์กับแอดเดรสที่แน่นอน

ตัวอย่าง ตารางเช่น

ตำแหน่ง	ข้อมูล	ข้อมูลเลขฐานสิบ
60	00	0 (0^2)
61	01	1 (1^2)
62	04	4 (2^2)
63	09	9 (3^2)
64	10	16 (4^2)
65	19	25 (5^2)
66	24	36 (6^2)
67	31	49(7^2)

การมองหาตารางเราก็ใช้วิธีสร้างอินเดคเช่น ต้องการรู้ว่าผลยกกำลัง 2 ของตัวเลข 3 มีค่าเท่าไร ก็ย่อมหาได้จากตาราง ลักษณะโปรแกรมที่ใช้หาจะเป็นดังนี้

```
00    3A           LAD    40H
01    40
02    00
03    6F           MOV    L,A
04    26           MVI    H,0
05    00
06    11           LXI    D,60H
07    60
08    00
09    19           DAD    D
0A    7E           MOV    A,M
0B    32           STA    41H
0C    41
0D    00
0E    C3    HEAR   JMP    HEAR
```

การสร้างตารางเป็นเทคนิคในการเขียนโปรแกรมที่ดีมากวิธีหนึ่งและเราสามารถนำมาใช้ได้ค่อนข้างมาก เช่น สร้างตารางค่าของ sin ก็เรียนกมาจากตารางโดยทราบว่าที่แอดเดรสเท่าไร่เป็นค่าไซน์มุมใด หรือในบางกรณีเราอาจสร้างตารางในการใช้ถอดรหัสบางอย่าง เช่น ใช้ถอดรหัสในภาคแสดงด้วย LED เจ็ดส่วนให้เป็นตัวอักษรพิเศษบางอย่างขึ้นเองได้โดยไม่ต้องอาศัยวงจรทางฮาร์ดแวร์ในการถอดรหัสเช่น ให้รหัส 7 ส่วนเป็นอักษร A-F ดังนี้ ABCDEF เป็นต้น

ในส่วนของโปรแกรมโมนิเตอร์ก็ดี โปรแกรมการแปลรหัสแอสเซมเบลอร์ก็ดี เรามักใช้วิธีการสร้างตารางเช่นว่านี้เพราะจะทำให้ประหยัดเนื้อที่ในหน่วยความจำลง

<u>คำสั่งเทียมที่ใช้ในการเขียนโปรแกรมภาษาแอสเซมบลี้ (Pesudo Assembler)</u>

ในการเขียนภาษาแอสเซมบลี้จะต้องเกี่ยวข้องกับตำแหน่งต่าง ๆในส่วนของโปรแกรม และถึงแม้ว่าเราจะหลีกเลี่ยงลักษณะของคำสั่งในภาษาเครื่องมาเขียนด้วยนีโมนิคส์แล้วก็ตาม ความยุ่งยากในการกำหนดค่าหรือการสร้างนิยามของตัวแปรก็เกิดขึ้น เราจึงหาวิธีการทำจะทำให้การเขียนโปรแกรมเป็นไปได้ง่ายยิ่งขึ้น ด้วยการกำหนดคำสั่งใหม่เพิ่มเติมขึ้นมา คำสั่งที่กำหนดขึ้นมา ... คำสั่งเช่นว่านี้จึงเป็นคำสั่งเทียม (Pseudo Instruction)

1. ORG ย่อมาจากคำว่า ORIGIN

คำสั่งนี้ไว้สำหรับให้ผู้โปรแกรมมากำหนดแอดเดรสของตัวแปรหรือโปรแกรมย่อยว่าอยู่ที่แอดเดรสใด เช่น

```
ORG    100
MVI  SP, DAT
```

ในที่นี้หมายถึงว่าเราเริ่มต้นโปรแกรมที่แอดเดรส 100 หรือ คำสั่ง MVI อยู่ที่แอดเดรส 100 นั่นเอง

ORG อาจจะใช้กำหนดอยู่ในรูปของตัวแปรก็ได้เช่น

```
ORG  RESET
MVI SP, DAT
```

ในที่นี้ค่า RESET จะต้องถูกนิยามค่ามาก่อน

2. EQU มาจากคำว่า EQUATE

คำสั่งนี้เป็นการกำหนดค่าให้กับตัวแปรนั่นเอง ลักษณะของการใช้จะเป็นดังนี้

```
RESET EQU 00
      ORG RESET
      MVI SP, DAT
```

ในกรณีนี้เป็นการกำหนดค่า FINAL ให้เท่ากับ TTY

3. DS มาจากคำว่า DATA STORAGE

คำสั่งนี้เป็นการกำหนดค่าว่าจะเก็บไว้ในอะเรย์ของหน่วยความจำจำนวนกี่ไบท์นั่นเอง

```
TEMP  EQU  00
      ORG   TEMP
MAN    DS  2
```

มีความหมายว่าตัวแปร MAN จะเก็บไว้ในหน่วยความจำ 2 ไบท์ คือที่แอดเดรส 00 กับ 01

4. SESERVE ลักษณะการใช้เหมือนกับ DS นั่นเองคือ จะเป็นการกำหนดพื้นที่ของหน่วยความจำขึ้นมาซึ่งอาจจะใช้สำหรับเป็นกลุ่มข้อมูลเป็นตาราง หรือสแตค ลักษณะวิธีใช้จะเป็นดังนี้

```
ORG 3000
BUF1 RESERVE 100
```

ในที่นี้ BUF 1 จะประกอบด้วยกลุ่มหน่วยความจำจากแอดเดรส 3000 นับไปอีก 100 ที่เรียงกันไป

การสร้างลูป (loop) ในโปรแกรม

ในภาษาชั้นสูงเราสามารถกำหนดการทำงานเป็นลูปของโปรแกรมได้ง่ายเช่น ในภาษา BASIC เรากำหนดการบวกข้อมูล จาก 1-10 ได้ดังนี้

```
10          J   = 0
20          FOR I= 1 TO 10 STEP 1
30          J    = I + J
40          NEXTI
50          ANS   =  J
```

หรืออีกกรณีภาษา FORTRAN เรามี DO กำหนด LOOP หรือการทำงานเป็นลูปได้ง่าย เพียงแต่บอกจำนวนลูปที่ต้องการจะทำตามกำหนด

ในส่วนของภาษาแอสเซมบลี้นั้นการกำหนดลูปเราก็ทำได้เช่นเดียวกันแต่จะต้องมีลักษณะบางอย่างที่ยุ่งยากกว่า เพราะที่เราจะต้องกำหนดในโปรแกรมประกอบด้วยส่วนสำคัญสี่ส่วนคือ

1. ส่วนการกำหนดค่าเริ่มต้น เช่นการกำหนดจำนวนที่ต้องการโดยการกำหนดขึ้นที่ส่วนของค่าในเคาว์เตอร์ที่เราต้องกำหนดค่าตัวแปรเริ่มต้นที่เราต้องการ

2. ส่วนของการประมวลผลจากข้อมูลที่เราต้องการ

3. ส่วนควบคุมลูป ซึ่งเป็นส่วนหนึ่งของการปรับค่าในเคาว์เตอร์เพื่อนับว่าครบตามจำนวนแล้วหรือยัง

4. ส่วนสรุปหรือเก็บผลลัพท์ที่ต้องการ

เพื่อเข้าใจในเรื่องในเรื่องโปรแกรมได้ดียิ่งขึ้นดังตัวอย่างต่อไปนี้

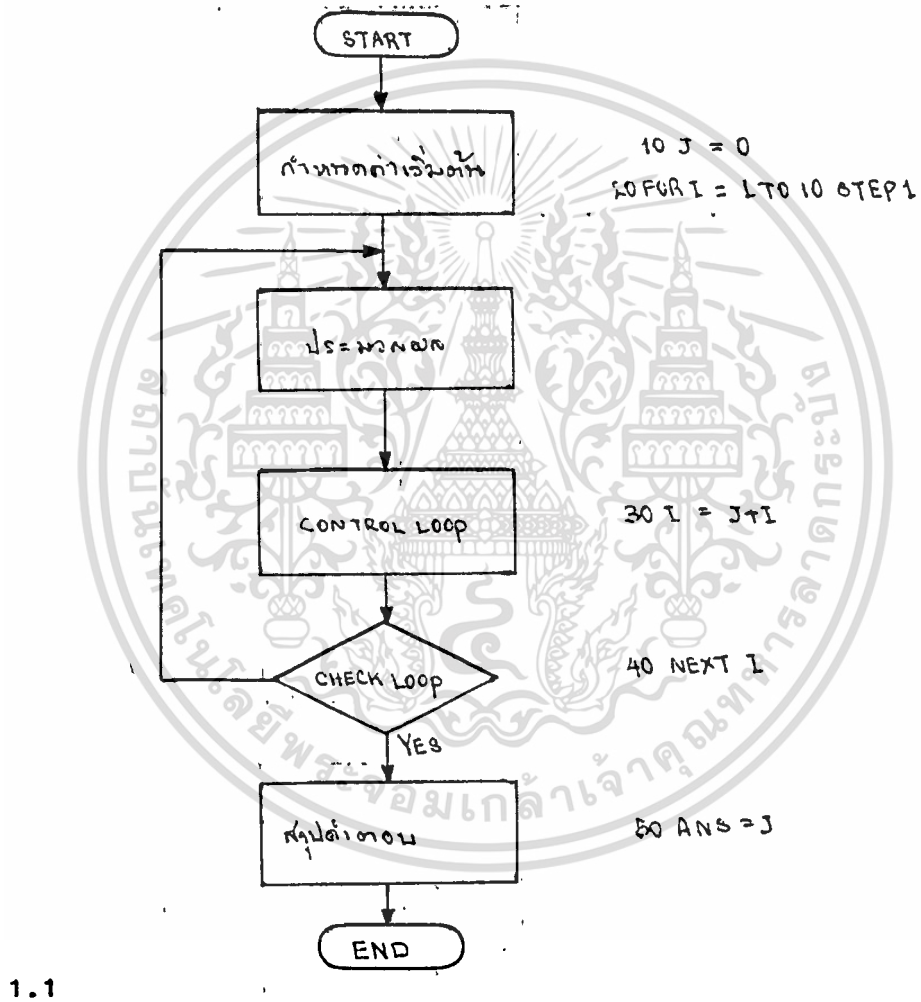
จงเขียนโปรแกรม แสดง การบวกกันของข้อมูลโดยจำนวนครั้งที่ใช้ในการบวกจะเก็บไว้ในหน่วยความจำครั้ง เก็บผลลัพท์ที่ได้ไว้ที่หน่วยความจำตำแหน่ง

หมายเหตุ เพื่อลดข้อยุ่งยากการเขียนรหัสภาษาเครื่องฐานสิบหกจึงขอเขียนโปรแกรมด้วยสัญญลักษณ์ทางนิโมนิคแต่ละเพียงอย่างเดียว

```
LXI   H, 41 H       ; COUNT  = LENGTH OFSERIES  OF NUMBER
MOV   B, M
SUB   A
SUMD  INX  H
```

```
ADD     M       ;SUM  = SUM + DATA
DCR     B
JNZ     SUMD    ; CHECK  NO  OF  LOOP
STA     40 H    ; STORE  SUM
HLT
```

จากกรณีนี้เรามาดูกันที่ผังงานจะเป็นดังรูปที่ 1.1

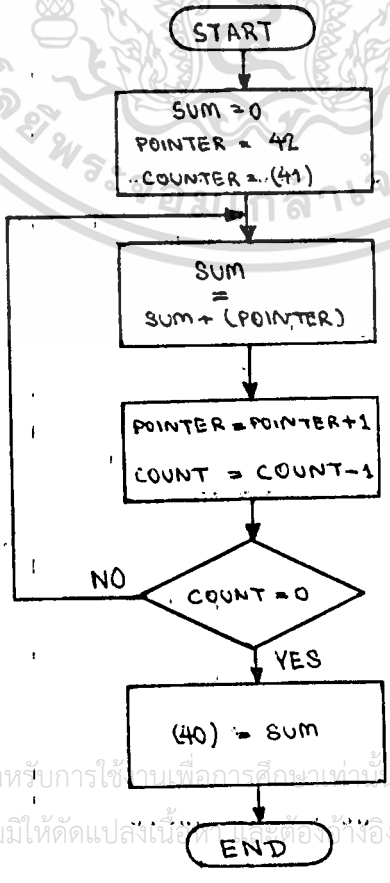


รูปที่ 1.1

ในการบวกตัวเลขหลายๆ ไบท์นั้นผลบวกอาจจะเกินกว่า 8 บิท ได้ เช่น C8+FA+96 จะมีค่าเป็น 0258 H ซึ่งมีค่ามากกว่า 8 บิท จากกรณีเช่นนี้เราดัดแปลงโปรแกรมที่กล่าวถึงแล้วให้เป็นได้ดังรูปที่

จากตัวอย่างข้างบนเราสมมติว่าจำนวนลูปของการบวกกำหนดด้วยหน่วยความจำตำแหน่งที่ 4216 และข้อมูลเราเริ่มบวกกำหนดด้วยหน่วยความจำตำแหน่งที่ 4316 ส่วน SUMH เก็บที่ 4116

```
LXI          H,  42H
MOV     B,M             ; COUNT  = LENGTH  OF SERIES
SUB     A               ; LSB OF SUM   = 0
MOV     C,A             ; MSB OF SUM   =0
DSUMD INX  H
ADD     M               ; SUM  = SUM+ DATA
JNC    CHCNT
INR    C                ; ADD CARRY  TO MSB OF SUM
CHCNT DCR B
JNZ   D SUMD
LXI   H, 40 H           ; STORE LSB OF SUM
MOV   M,A
INX   H                 ; STORE MSB OF SUM
MOV     M,C
HLT
```



รูปที่ 1.2

ตัวอย่าง สมมติว่ามีข้อมูลอยู่หนึ่งบล็อก ส่วนของความยาวบล็อกเรากำหนดไว้ด้วยความหน่วยจำตำแหน่งที่ 4116 ส่วนจุดเริ่มต้นของบล็อกอยู่ที่ 4216 จงหาค่าที่มากที่สุดของข้อมูลที่เก็บไว้ในหน่วยความจำบล็อกที่กำหนด

จากกรณีนี้เราเขียนเป็นผังงานได้ดังนี้

สำหรับตัวโปรแกรมตามผังงานเขียนได้เป็น

```
LXI   H, 41 H    ; POINT TO COUNT
MOV  B,M         ; COUNT =UNMBER OF ELEMENTS
SUB A            ; MAXIMUM  =0
NEXT  INX  H
CMP  M           ; IS NEXT ELEMENT MAXIMUM
INC DECN
MOV A,M          ; YES REPLANCE MAXIMUM
DECN DCR B
JNZ   NEXT
STA  40 H        ; SAVE  MAXIMUM
HLT
```

<u>การกำหนดผังงาน</u>

<u>การออกแบบระบบซอร์ฟแวร์</u>

การออกแบบระบบซอร์ฟแวร์เหมือนกับการออกแบบวงจรอิเลคทรอนิคส์หรือเครื่องจักรกลทั่วๆ ไป ซึ่งจะมีเริ่มต้นด้วยการกำหนดขอบข่ายของปัญหา ที่ต้องการจะแก้ไขและรายละเอียดของข้อมูลที่ต้องการหรือผลลัพท์ ในทางซอร์ฟแวร์เราจำเป็นต้องกำหนดโครงสร้างของโปรแกรมโดยยึดหลักการดังต่อไปนี้

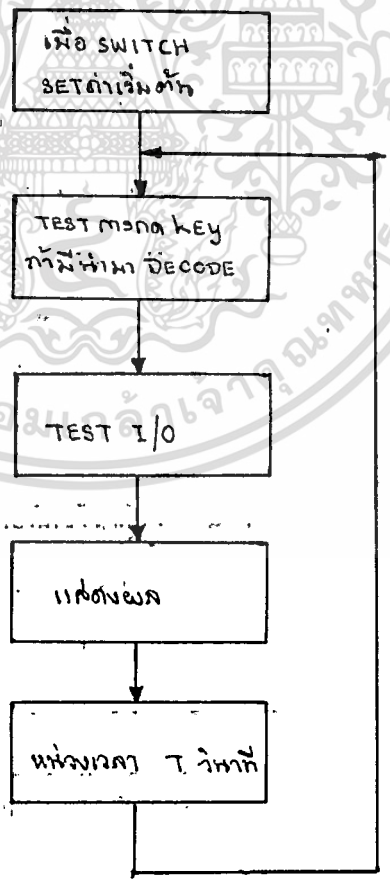
1. แจกแจง และตีความหมายของปัญหารวมทั้งคิดแนวทางหรือวิธีที่จะใช้ในการแก้ปัญหานั้น ๆ
2. กำหนดผังของโปรแกรม
3. เขียนรายละเอียดส่วนย่อยในโปรแกรม

จากกรณีนี้เราลองยกตัวอย่างระบบซอร์ฟแวร์ง่าย ๆ ที่ใช้ไมโครคอมพิวเตอร์ให้ดูสักตัวอย่างหนึ่ง โดยสมมติว่าเรามีซีพียูที่ติดต่อกับอินพุทหลายตัว และอินพุทตัวหนึ่งอาจจะเป็นคีย์บอร์ด และเอาท์พุทตัวหนึ่งอาจจะเป็นภาคแสดง LED สิ่งที่เราต้องการคือ ในการคีย์โปรแกรมหรือข้อมูลผ่านทางคีย์บอร์ดนั้นจะให้ผลของปัญหาและคิดหาวิธีได้แล้วก็จะมาเขียนผังงานตามวิธีที่เลือกไว้

<u>วิธี พอลลิ่ง POLLING</u>

จากกรณีผังงานรูปที่ นี้ จะเห็นว่าจะต้องให้ซีพียูคอยตรวจดูเป็นระยะ ๆ ว่าคีย์บอร์ดมีการคีย์อะไรมาหรือไม่ ถ้าไม่มีก็ผ่านไป ทดสอบ I/O ตัวอื่นต่อไป ดังนั้นซีพียูจะต้องเสียเวลามาคอยวนอ่านคีย์บอร์ดอยู่เสมอ ๆ

<u>วิธีการอินเตอร์รัปท์</u> ในกรณีนี้ผังงานอาจจะลดข้อยุ่งยากทางซอร์ฟแวร์บ้างแต่จะยุ่งยากในวงจรอินเตอร์รัปท์อยู่บ้าง ลักษณะของการเขียนเป็นผังงานของการกระทำในกรณีนี้ดังนี้

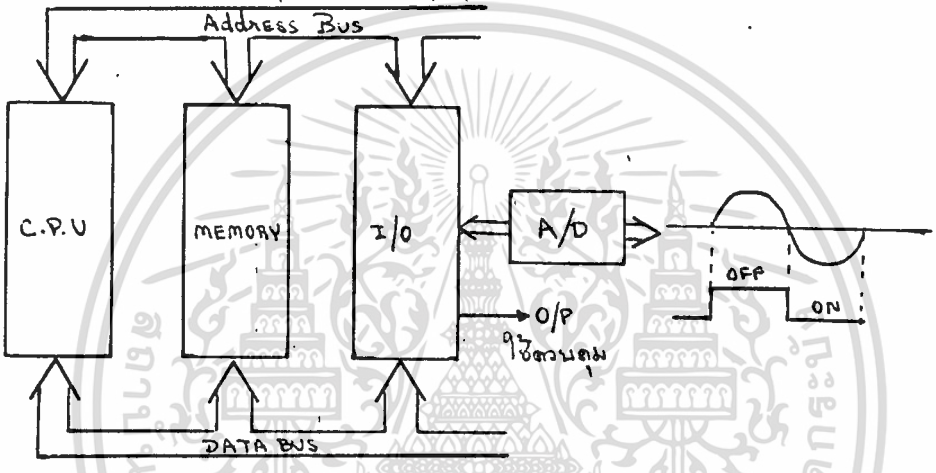


รูปที่ 1.3

ในการอินเตอร์รัฟท์ซีพียูจะต้องสนองทันทีที่มีสัญญาณอินเตอร์รัฟท์เข้ามา เรามีเทคนิครายละเอียดเกี่ยวกับอินเตอร์รัฟท์อีกมาก ซึ่งจะได้อธิบายให้เห็นในบทต่อไป

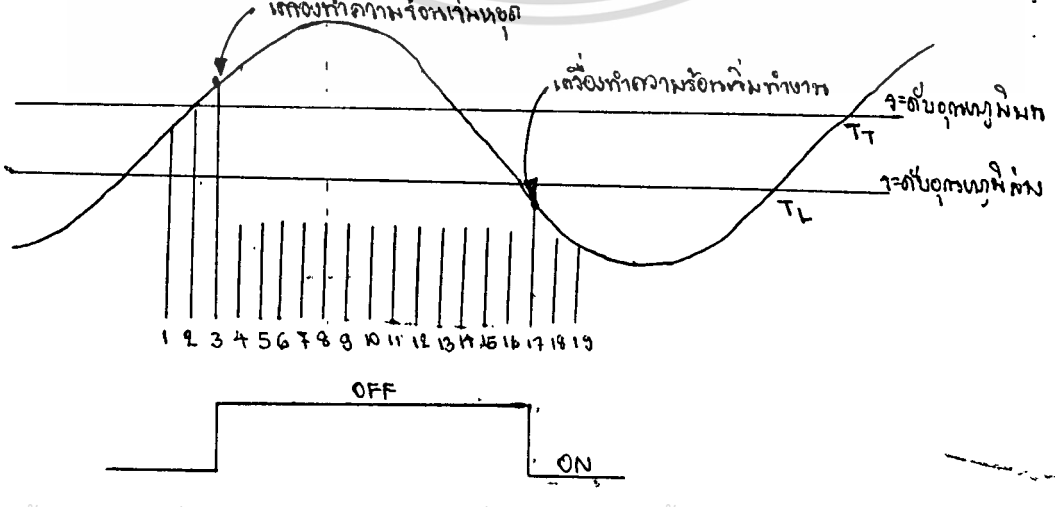
<u>ตัวอย่างการวางระบบซอร์ฟแวร์</u>

เพื่อเป็นแนวทางอย่างง่ายจึงขอยกตัวอย่างง่ายให้เป็นแนวทางในการเข้าใจ ในที่นี้สมมติว่าต้องการระบบซอร์ฟแวร์ที่ใช้ควบคุมอุณหภูมิของโรงงานอุตสาหกรรมแห่งหนึ่งระบบคอมพิวเตอร์เป็นระบบคอมพิวเตอร์เป็นดังรูป



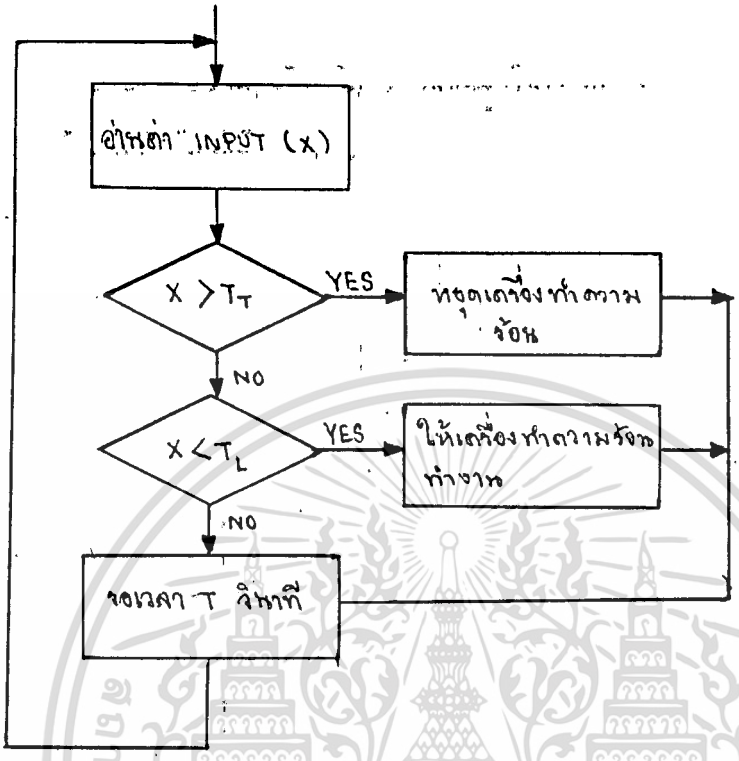
รูปที่ 1.4

สิ่งที่ต้องการคือลักษณะของสัญญาณอินพุทและเอาท์พุทที่ต่อยู่กับไมโครคอมพิวเตอร์โดยอินพุทเป็นค่าอุณหภูมิที่เราวัดได้ส่วนเอาท์พุทคือ สัญญาณที่จะไปควบคุมการทำงานของเครื่องทำความร้อน ลักษณะของสัญญาณทั้งสองสัมพันธ์กันดังนี้



รูปที่ 1.5

จากจุดมุ่งหมายที่เราได้วางไว้เรานำมาเขียนผังงานในการทำงานได้ดังนี้



รูปที่ 1.6

ในส่วนของการรอเวลานั้นเราอาจใช้วิธีการกำหนดลูปให้เกิดการทำงานโดยคำนวณหาจำนวนลูปที่พอดี ทำให้ระยะที่ได้ตามต้องการ ลักษณะเช่นนี้เราเขียนเป็นผังงานในส่วนการรอเวลาได้ดังนี้

การเขียนโปรแกรมที่เกี่ยวข้องกับข้อมูลเป็นตัวอักษรหรือตัวเลข

บ่อยครั้งที่เดียวที่เราอดที่จะแปลดใจไม่ได้ว่า การทำงานของโปรแกรมก็ดี การใช้งานก็ดีมักจะใช้ลักษณะคำสั่งที่ THEN Y= 3 ; NEXT I ทั้งๆ ที่การทำงานของเครื่องคอมพิวเตอร์จะใช้รหัสคำสั่งในรูปภาษาเครื่องที่เป็นเลขไบนารี่ การที่เราแทนลักษณะคำสั่งหรือข้อความแล้วเราเข้าใจได้ดีเป็นสิ่งที่นิยมทำกันมากเช่น ADD หมายถึงการบวก

เราทราบแล้วว่าตัวอักษรในภาษาอังกฤษนั้นสามารถแทนเป็นรหัสตัวเลขไบนารี่ได้ ซึ่งขึ้นอยู่กับมาตรฐานที่ใช้ได้ และมาตรฐานได้มากถึง $2^7 = 128$ ตัวอักษร

ลักษณะเด่นของรหัส ASCII นั้นเป็นสิ่งที่เราจะต้องเข้าใจในหลักการทั่วๆ ไปก่อนเช่น ถ้าต้องการจะพิมพ์ตัวเลข 9 ที่เครื่องพิมพ์ไมโครคอมพิวเตอร์จะต้องส่งรหัส 3916 มาให้เครื่องพิมพ์ นอกจากรหัสที่ใช้เป็นตัวเลขและตัวอักษรแล้ว ยังมีรหัสอีกกลุ่มหนึ่งที่ใช้เป็นรหัสในการควบคุมระบบโดยรหัสเหล่านี้เป็นที่ยอมรับกันอยู่แล้ว เช่น 0A16 หมายถึงการเลื่อนบรรทัด (LF) 0D หมายถึงรหัสแทน carriage return (CR) รหัส 20 หมายถึง space (SP) เป็นต้น

เมื่อระบบที่ใช้มีรหัสในการควบคุมที่ยอมรับแล้ว การกำหนดการทำงานระบบจึงทำให้ง่ายและมีประสิทธิภาพได้ดียิ่งขึ้น เช่น เมื่อต้องการให้ตัวอักษรบนจอ CRT ขึ้นบรรทัดใหม่เครื่องคอมพิวเตอร์ก็เพียงแต่ส่งรหัสตัวอักษร CR ไปให้เท่านั้น

โดยปกติการวางรูปของข้อมูลตัวอักษรหรือตัวเลขก็ดีมักผูกรวมกันเป็นกลุ่ม ๆ ในกรณีตัวเลขก็จะรวมกลุ่มกันเป็นตัวเลขอินทีเจอร์ (integer) เลขจำนวนจริง (real) เลขโฟลตติ้งพอยท์ (floating point) สำหรับกลุ่มตัวเลขอินทีเจอร์หรือจำนวนเต็มอาจใช้จำนวนจริงอาจต้องใช้ 3-4 ไบท์รวมกันและตัวเลขโฟลตติ้งพอยท์ ก็อาจรวมข้อมูลถึง 8 ไบท์ เป็นต้น

ในกรณีของตัวอักษรเราก็จะผูกรวมกันเป็นข้อความที่เรียกว่าสติง เช่น ข้อความว่า GOOD รวมกันก็เป็นสติง เพื่อให้เข้าใจเกี่ยวกับการนำรหัสข้อมูล ASCII มาใช้เป็นประโยชน์จึงขอยกตัวอย่างดังนี้

ตัวอย่าง ในการถ่ายข้อมูลจากหน่วยความจำไปยังเครื่องพิมพ์นั้นต้องการนับว่าจำนวนสตริงของข้อมูลที่ส่งไปให้เครื่องมีจำนวนตัวอักษรทั้งหมดกี่ตัวโดยข้อมูลที่จะส่งเริ่มต้นตั้งแต่แอดเดรส 4116 เป็นต้นไป การจบของสตริงจะปิดท้ายด้วยสัญญาณ CR ที่นับได้ให้นำหน้าไปเก็บไว้ในแอดเดรส 4016

จากผังงานเราจะเขียนโปรแกรมตามผังงานได้

```
LXI  H,  41H      ; POINTER  = START  OF STRING
MVI  B, 0         ; LENGTH =0
CHKCR CMP M       ; IS CHARACTER CR?
JZ  DONE          ; YES END OF STRING
INR  B            ; NO, ADD 1 TO LENGTH
INX  H
JMP CHKCR         ; EXAMINE NEXT CHARAC TER
DONE MOV A,B
STA 40H           ; SAVE STRING LENGTH HERE JMP HERE
```

ในกรณีของอุปกรณ์เปอริเฟอรัลเช่น เครื่องพิมพ์จะรับจากคอมพิวเตอร์มานิมพ์ ส่วนตัวและรหัสควบคุมเครื่องนั้นเครื่องจะรู้เองและจะไม่มีการพิมพ์แต่จะรับรู้และกระทำตามคำสั่งรหัสควบคุม

การเปรียบเทียบข้อมูลด้วยคำสั่ง CMP นั้น ผลที่ได้ในขณะที่เท่ากันจะทำให้แฟลกศูนย์ได้รับการเซท ครั้นเราตรวจสอบ JZ มันก็จะทำงานโดยกระโดดต่อไปได้

ลักษณะของโปรแกรมดังกล่าวนี้สามารถเขียนผังงานได้อีกแบบหนึ่งเป็น

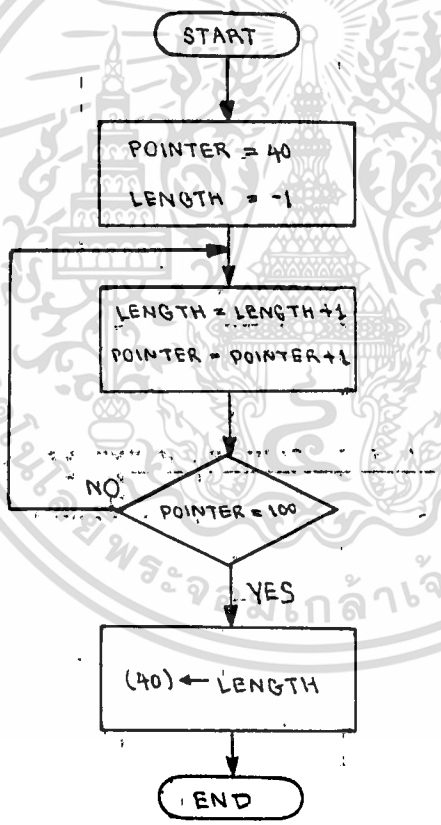
```
LXI  H,40H  ; POINTER = BYTE BEFORE STRING
MVI B OFFH  ; LENGTH =-1
MIV A, oDH
CHKCR INX H
INR B       ; ADD 1 TO LENGTH
CMP M       ; IS CHARACTER 'CR'?
JNZ CHKCR   ; NO, KEEP COUNTING
MOV  A,B
STA 40H     ; YES, SAVE STRING LENGTH
              HEAR JMP HERE
```

ลักษณะการเปรียบเทียบทำการตรวจสอบบล๊อก
ในตัวอย่างนี้เป็นลักษณะที่นำไปประยุกต์ใช้โหลดข้อมูล
จากเทปคาสเซทหน่วยความจำ ซึ่งมาตรฐานคือ

INTEL HEX FORMAT จะใช้ตัวอักษรใน ASCII คือ CR นี้เป็นตัวกำหนดว่าข้อมูลในบล๊อกนั้นแล้ว

<u>ตัวอย่างโปรแกรมแทนเลขฐานศูนย์ที่อยู่ข้างหน้าด้วยแบลงค์</u>

การเติมแบลงค์แทนเลขฐานศูนย์สำหรับตัวเลขนั้นมีความจำเป็นในทางผลลัพท์ทางคณิตศาสตร์ ในกรณีของเครื่องพิมพ์เลข 0 อยู่ข้างหน้า ต่อไปนี้เป็นตัวอย่างที่ยกขึ้นมาให้เห็นจริงโดยกำหนดว่าสตริงของตัวเลขนี้เริ่มต้นที่หน่วยความจำแอดเดรส 4016 โดยที่จำนวนความยาวของสตริงจะเก็บไว้ที่แอดเดรส 4016 ลักษณะของผังงานจะเขียนได้เป็น



รูปที่ 1.7

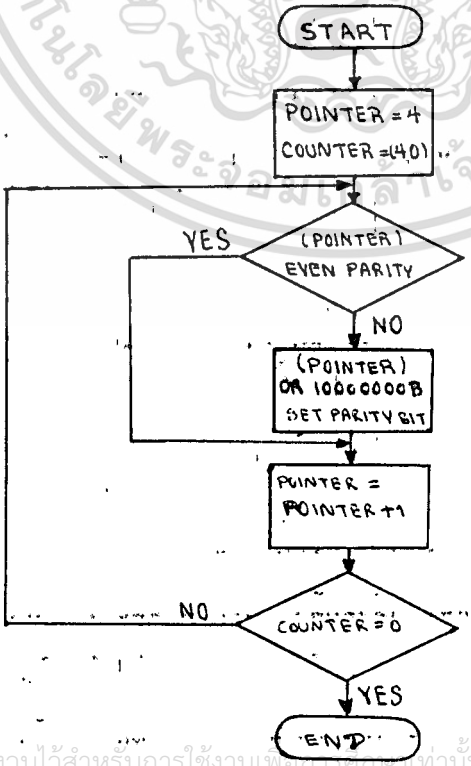
จากผังงานเราเขียนเป็นโปรแกรมได้เป็น

```
      LXI   H,40H
      MOV   B,M    ;   COUNT   =  STRING  LENGTH
      MVI   A, 0   ;   GET  ASCII  0  FOR  COMPARISION
CHKZ: INX   H
      CMP    M     ;   NO, THROUGH
      JNZ   DONE   ;   REPLACE-LEADING ZERO WITH BLANK
      DCR   B      ;   HAVE  ALL  DIGITS  BEEN  EXAMINED?
      JNZ   CHKZ   ;   NO.GO EXAMINE  NEXT  HLT
```

<u>ตัวอย่างการเพิ่มบิทพาริตี้ ให้กับตัวอักษรใน ASCII</u>

ในการรับส่งข้อมูลทางคอมพิวเตอร์ทั่วไปมักจะมีการผิดเพี้ยนไปจากข้อมูลเดิม การผิดนั้นจะมีโอกาสที่ผิดพลาดวิธีหนึ่งที่ง่ายก็ได้แก่การเพิ่มบิทของข้อมูลในแง่ตัวอักษรเดิมผิดไป แต่บิทที่เติมจะเป็น"0" "1" ก็ได้ ที่พอดีทำให้ผลบวกของตัวเลข 1 ก็ได้

ในตัวอย่างนี้สมมติว่าสตริงของตัวอักษร ASCII ชนิด 7 บิท สิ่งที่เราต้องการคือให้ตัวอักษรทุกตัวจะเพิ่มบิทที่เป็นพาริตี้เพื่อให้แสดงว่าพาริตี้เป็นคู่ จากลักษณะเช่นว่านี้เขียนผังงานได้เป็นดังนี้



รูปที่ 1.8

เมื่อเขียนโปรแกรมจะได้

```
LXI   H,40H
MOV   B,M       ; GET STRING LENGTH
MVI   C, 10000000;GET PARITY BIT OF 1
SETPR  INX   H
MOV A,M         ; GET A CHARACTER
ORA  C          ; SET PARITY  BIT TO 1
JPO  CHCNT      ; IS PARITY NOW EVEN?
MOV  M,A        ; YES, SAVE CHARACTER WITH EVEN PARITY
CHCNT  DCR  B
JNZ  SETPR
HLT
```

จากลักษณะที่สำคัญของโปรแกรมนี้อยู่ที่การกำหนดค่า 10000000 ขึ้นรีจิสเตอร์ C เพื่อนำมา OR นี้จะเป็นผลทำให้บิทพาริตี้เป็น 1 เสมอไม่ว่ากรณีใด ๆ ดังนั้นเราจึงมีข้อมูลมาทดสอบที่รีจิสเตอร์ A มีค่าเป็น 1 ที่บิทแรกสุด ครั้นเมื่อเราทดสอบด้วยคำสั่ง JPO หรือ JUMP IF ODD PARITY นั่นคือข้อมูลที่อยู่รีจิสเตอร์ A เป็นพาริตี้คู่ก็คือค่าเดิมในหน่วยความจำของมันนั่นเองที่มีค่าพาริตี้คี่ ซึ่งค่าพาริตี้คู่นี้ ก็คือค่าเดิมในหน่วยความจำของมันนั่นเองที่มีค่าพาริตี้ถูกต้องแล้ว ถ้าในรีจิสเตอร์ A มาเก็บไว้แทนหน่วยความจำ

เพื่อให้เข้าใจถึงขบวนการดังข้างบน เราสมมุติว่าเดิมหน่วยความจำในแอดเดรสต่างๆ เก็บข้อมูลไว้ดังนี้

(40) = 06

(41) = 31

(42) = 32

(43) = 33

(44) = 34

(45) = 35

(46) = 36

หลังจากทำการเอ็กซีคิ้วแล้วผลที่ได้จะเป็นดังนี้

(41) = B1

(42) = B2

(43) = 33

(44) = B4

(45) = 35

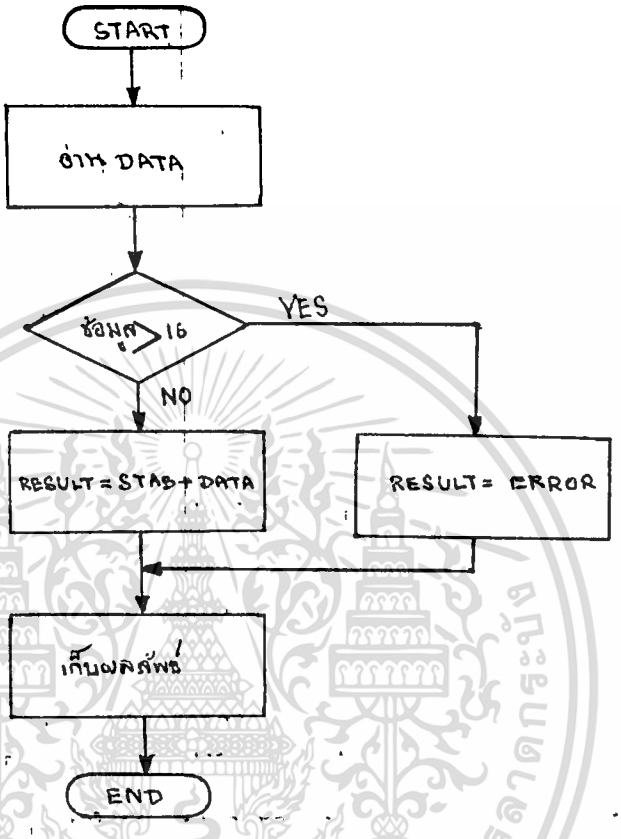
(46) = 36

การเขียนโปรแกรมเพื่อเปลี่ยนรหัสจากตัวเลขฐานสิบหกให้แสดงด้วยผลภาคแสดง 7 ส่วน

ในการเขียนโปรแกรมเพื่อควบคุมภาคแสดง LED 7 ส่วน มีเทคนิคในการเขียนหลายแบบด้วยกัน ทั้งนี้มักจะมีส่วนของวงจรทางอาร์ดแวร์มาเกี่ยวข้องด้วยเสมอ แต่อย่างไรก็ตามที่มักใช้กันมากอย่างหนึ่งก็คือกำหนดข้อมูลในแต่ละบิทในแต่ละบิทแทนแต่ละส่วนของ LED ดังนั้น LED 7 ก็จะแทนในส่วนของข้อมูบเหล่านั้น แล้วจึงใช้โปรแกรมมองหาข้อมูลตามที่ต้องการส่งไปยังภาคแสดง วิธีการเช่นนี้เรียก Table Look Up ขอให้พิจารณาหลักการสร้างตารางดังนี้

ตัวเลข	รหัส
0	3F
1	06
2	5B
3	4F
4	66
5	6D
6	7D
7	07
8	7F
9	6F
A	77
B	1F
C	4E
D	3D

การติดสว่างของ LED จะเป็นไปตามลักษณะบิทโดยบิทที่แสดงค่า 1 จะเป็นบิทที่ติดสว่าง เมื่อสร้างตารางไว้เป็นข้อมูลในหน่วยความจำแล้ว เราก็นำข้อมูลที่อ่านมาหาดูว่าตรงช่องตารางใด ก็เอาข้อมูลนั้นออกมาก็จะได้รหัสตามภาคแสดง 7 ส่วน ซึ่งเราจะเขียนเป็นผังงานได้



รูปที่ 1.9

```
START  MVI  B, ERR   ; GET ERROR CODE
  IN   PORT1         ; READ INPUT
  CPI  16            ; IS DATA > 16
  JNC  DONE          ; YES DONE
  LXI  H, STAB       ; LOAD STARTING TABLE  ADDRESS
  MOV  C,A           ; MAKE DATA  INTO A 16 BIT INDES
  DAD  BB            ; FIND ADDRESS OF TABLE
  MOV  B,M           ; LOOK UP TABLE
DONE  MOV A,B
  OUT  PORT 2
```

```
STAB    DB  3FJ. 06H, 4F, 66H,- TABLE CONTENT
        DB  6DH, 7DH,07H,7FH,6FH
        DB  77H, 1FH, 4EH, 3DH, 4FH
        DB  47 H
ERR     DB  55H         ; ERROR CODE
```

จากกรณีนี้จะเห็นว่าในการทำงานกำหนดข้อมูล (DB = defined byte) นั้นแอดเดรสสุดท้ายคือ 55 H นั้นเป็นรหัสของตัวบอกข้อผิดพลาดในกรณีที่มีรหัสเข้ามามากกว่า 15 ลักษณะเช่นนี้เราสามารถกำหนดอักษรต่างๆ จากภาคแสดง 7 ส่วนได้เป็นอย่างดี โดยการสร้างตารางของตัวอักษรเหล่านั้นขึ้นมา และสมมติว่าเมื่อเรากดคีย์บอร์ดของตัวอักษรตัวหนึ่งเข้ามา ซีพียูจะอ่านข้อมูลเข้ามาแล้วตีความมองหาข้อมูลจากตารางแล้วส่งผลลัพท์ไปที่ภาคแสดงตามที่เราคีย์เข้ามาได้

<u>การเขียนภาษาแอสเซมบลี้ในทางคณิตศาสตร์</u>

โดยหลักการทั่วไปแล้วการเขียนภาษาแอสเซมบลี้ไม่เหมาะกับการคำนวณทางาคณิตศาสตร์ที่ยุ่งยากซับซ้อน เพราะการทำงานทางคณิตศาสตร์เรามักจะมองใกล้ในแง่ของภาษาเครื่อง ทำให้เราต้องยุ่งยากต่อตัวเลขไบนารี่เป็นจำนวนมาก แต่อย่างไรก็ตามในภาษาชั้นสูงที่เหมาะในการทำงานคำนวณ เช่น BASIC FORTRAN ก็อาศัยโครงสร้างการคำนวนพื้นฐานเหล่านี้

<u>การบวกตัวเลขที่มีความเชื่อถือได้หลายหลัก</u>

การทำงานของ 8080 จะกระทำข้อมูลที่ละไบท์ (8 บิท) ถ้าพิจารณาในแง่ตัวเลขจะเห็นว่าเป็นจำนวนเลขที่น้อยมาก แต่การคำนวณที่เราใช้นั้นอาจมีจำนวนตัวเลขฐานสิบได้มากกว่า 10 ตัว ดังนั้นเราทำได้โดยไม่ยากนักโดยพิจารณาจากตัวอย่างต่อไปนี้

เมื่อเราเขียนโปรแกรมจะได้ดังนี้

```
        LDA  30 H       ; COUNT =LENGTH  OF STRINGS (IN BYTES)
        MOV  B,A
        LXI H,41 H      ; START POINTER 1 AT FIRST WORD OF STRING1
        LXI D, 61H      ; START POINTER 2 AT FIRST WORD OF STRING2
        AND A           ; CLEAR CARRY TO START
ADDW LDAX D             ; GET WORD FROM STRING 2
        ADC  M          ; ADD WORD FROM STRING 1
        MOV  M,A        ; STORE  RESULT
        INX  D
        INX  H
        DCR  B
        JNZ  ADDW
```

การบวกในลักษณะนี้เราสามารถบวกต่อข้อมูลจำนวนกี่ไบท์ต่อกันก็ได้ แต่อย่างไรก็ตามในภาษาชั้นสูง เช่น FORTRAN ตัวเลขที่เป็นจำนวนจริง เรามักใช้เพียงจำนวน 4 ไบท์เท่านั้น ส่วนตัวเลขที่เป็นจำนวนเต็ม (integer) เราอาจลดลงมาเหลือเพียง 2 ไบท์ (16 บิท)

ในชุดคำสั่งของ 8080 มีคำสั่งที่ทำไว้สำหรับการบวกตัวเลข (BCD) รหัสฐานสิบได้ ลักษณะการบวกจะเป็นไปได้เหมือนเดิมโดยสมมติว่าแต่ละไบท์เป็นของข้อมูลจะเก็บตัวฐานสิบไว้สองหลัก เช่น

```
36701985+
12663459
49365444
```

การบวกเช่นนี้กระทำได้โดยไม่ยากเย็นอะไรเลย เราเพียงแต่ใช้วิธีการบวกตามที่เคยใช้มาแล้ว แล้วใช้คำสั่งแทรกประกอบ คือ คำสั่ง DAA ตามหลังคำสั่ง ADC เพื่อปรับผลของการบวกที่ได้ให้เป็นเลขฐานสิบเสียก่อน

การกระทำตามคำสั่ง DAA นั้นมีหลักการที่น่าสนใจอยู่ที่ควรแก่การรู้ คือ ในคำสั่งนี้จะใช้ส่วนของแฟลกตัวทด (auxillary carry flag) เป็นตัวแก้ไขตัวเลขในคำสั่ง DAA พอสรุปเป็นข้อ ๆได้ดังนี้

1. ในกรณีที่ผลรวมมาของตัวเลขมีค่าเกิน 9 คืออยู่ระหว่าง 10-15 ในกรณีนี้คำสั่ง DAA จะกระทำโดยการบวกค่าตัวเลข 0110 เพื่อแก้ตัวเลขดังกล่าวเช่น

```
    0101      (5)
+   1000      (8)
    1101      (D)
+   0110
0001 0011     (BCD = 13)
```

2. ในกรณีที่ผลบวกตัวเลขมีผลลัพท์ได้มากกว่า 16 ในกรณีนี้ค่าที่ปรากฏในตอนแรกจะมีค่าไม่เป็นไปตามข้อ 1 แต่ถ้าการทดปรากฏในแฟลตตัวทดช่วยหรือแฟลกตัวทด คำสั่ง DAA จะกระทำการปรับค่าด้วยการบวกตัวเลข 0110 เข้าด้วยการบวกตัวเลข เช่น

```
    1000 (8)
    1001 (9)
0001  0001  (BCD =11)
    + 0110
0001  0111  (BCD =17)
```

การคูณหารตัวเลขไบนารี่

ในไมโครโปรเซสเซอร์บางเบอร์ โดยเฉพาะกลุ่ม 16 บิท จะมีคำสั่งการคูณหารได้โดยเฉพาะกลุ่ม 16 บิทจะมีคำสั่งการคูณหารได้โดยตรง แต่สำหรับ 8080 แล้วการคูณหารจะไม่มี ดังนั้นหากเราต้องการใช้คำสั่งในการคูณหารเลขเราก็จำเป็นจะต้องเขียนโปรแกรมขึ้นเพื่อใช้ในการคูณหาร

อัลกอริทึมในการคูณ

ในการคูณตัวเลขไบนารี่ด้วยคอมพิวเตอร์นั้นก็เหมือนการคูณตัวเลขด้วยมือนั่นคือเรามีขั้นตอนประกอบอย่างไร เราก็อาศัยขั้นตอนมาใช้ในการคูณตามแบบอย่างเช่น

```
    11011011
*   10010011
    11011011
   11011011
  11011011
 11011011
1111011100001
```

จากผลคูณที่ได้ดังตัวอย่างข้างบนจะเห็นว่าการคูณก็คือการเลื่อนบิทมาทางซ้ายของตัวตั้ง แล้วบวกกันเข้าไปตามตำแหน่งเลขของตัวคูณ ดังนั้นเราสามารถวางเงื่อนไขของการคูณเป็นขั้นตอนต่าง ๆ ได้ดังนี้

ขั้นตอนที่ 1 เซทค่าเริ่มต้น

PRODUCT = 0

COUNTER = 8

ขั้นตอนที่ 2 เลื่อน PRODUCT ให้ตรงหอเหมาะ PRODUCT = 2x PRODUCT ถ้า lsb=0

ขั้นตอนที่ 3 เลื่อนตัวคูณ (MULTIPLIER) เพื่อให้บิทเลื่อนสู่แฟลกตัวทด MULTIPLIER = 2x MULTIPLIER

ขั้นตอนที่ 4 บวกตัวตั้ง (multiplicand) เข้ากับ PRODUCT ถ้าแฟลกตัวหดเท่ากับ 1

ขั้นตอนที่ 5 ลดค่า COUNTER แล้วตรวจสอบว่าเป็นศูนย์ COUNTER = COUNTER-1 ถ้า COUNTER ไม่เท่ากับ ศูนย์ ให้ไปทำขั้นตอนที่ สองใหม่

เพื่อให้เห็นอัลกอริทึมในการคูณได้ชัดเจนขอให้ดูลักษณะการทำการคูณของคอมพิวเตอร์เป็นขั้นๆ โดยสมมุติว่าตัวคูณคือ 61H และตัวตั้งคือ 6FH กรรมวิธีการคูณเป็นดังนี้

ขั้นเริ่มกำหนดค่า

PRODUCT	0000
MULITIPLIER	61
MULTIPLICAND	6F
COUNTER	8

หลังจากการทำครั้งแรกตามขั้นตอนที่ 2-5 ผลที่ได้จะเป็นดังนี้

PRODUCT	0000
MULTIPLIER	C2
MULTIPLICAND	6F
COUNTER	07
CARRY FORM MULTIPLIER	0

หลังจากทำการครั้งที่ 2 ผลที่ได้

PRODUCT 006F

MULTIPLIER 84

MULTIPLICAND 6F

COUNTER 05

CARRY FORM MULTIPLIER 1

หลังจากทำครั้งที่ 3 ผลที่ได้

PRODUCT 014D

MULTIPLIER 20

MULTIPLICAND 6F

COUNTER 03

CARRY FORM MULTIPLIER 0

หลังจากทำครั้งที่ 4

PRODUCT 029A

MULTIPLIER 10

MULTIPLICAND 6F

COUNTER 04

CARRY FORM MULTIPLIER 0

หลังจากทำครั้งที่ 5

PRODUCT 0534

MULTIPLIER 20

MULTIPLICAND 6F

COUNTER 03

CARRY FORM MULTIPLIER 0

หลังจากทำครั้งที่ 6

PRODUCT 0A68

MULTIPLIER 40

MULTIPLICAND 6F

หลังจากทำครั้งที่ 8

PRODUCT	2A0F
MULTIPLIER	00
MULTIPLICAND	6F
COUNTER	00
CARRY FORM MULTIPLIER	1

จากกรณีนี้ถ้าสมมติว่าต้องการคูณตัวเลขที่เก็บไว้ที่แอดเดรส 4016 กับแอดเดรส 4116 ผลลัพท์ที่ได้เก็บไว้ที่แอดเดรส 4216 และ 4316

<u>การตรวจสอบข้อมูลด้วยวิธีการทดสอบผลบวก (CHECK SUM)</u>

ในกรณีป้อนข้อมูลหรือโหลดข้อมูลจากที่หนึ่งไปเก็บยังอีกที่หนึ่งมักจะประสบปัญหาผิดเพี้ยนของข้อมูล เช่น ตัวเลข 43 ผู้ป้อนหรือป้อนเป็น 34 สลับกันในขณะกดตัวเลขเข้าไป ลักษณะการผิดเพี้ยนเช่นนี้ว่าควรจะมีการตรวจสอบที่ได้ผลดี วิธีหนึ่งที่มีการใช้กัน ข้อมูล 7260,3525 เป็นข้อมูล 2 ตัวที่ต่ออนุกรมกันมา แต่การรับจะหลายเป็น 2603 เป็นตัวเลขตัวหนึ่ง

วิธีการทดสอบแบบนี้จะอาศัยหลักการบวกค่าตัวเลขเหล่านี้ไม่ได้ ทั้งนี้เพราะจะทำให้เราตรวจสอบการสลับที่ไม่ได้ การตรวจสอบจึงต้องอาศัยเทคนิควิธีการช่วยเพิ่มเติม โดยการหาตัวเลขมาคูณหลักใดหลักหนึ่ง วิธีที่ใช้กันมากวิธีหนึ่งก็คือการคูณด้วย 2 ดังตัวอย่าง

ตัวเลข 54931

ผลของการทดสอบผลบวก = 5x2+4+9x2+3+2x2+1

= 40

ค่าของผลการทดสอบบวก = 0

วิธีการนี้เรามักจะเขียนขึ้นในโปรแกรมการโหลดบล๊อกข้อมูลลงเทปคาสเซท โดยแต่ละบล๊อกของข้อมูลตรวจสอบการทดสอบอีกครั้งหนึ่งว่าผลลัพท์ที่ได้ตรงกันหรือไม่ ตรงกันก็แสดงว่าการโหลดข้อมูลนั้นถูกต้อง

จากกรณีนี้เราลองยกตัวอย่างโปรแกรมการตรวจสอบผลบวกโดยข้อสมมติของการตรวจสอบข้อมูล 1 บล๊อกกโดยแอดเดรส 30 ผลลัพท์ของการตรวจสอบเก็บไว้ที่ตำแหน่ง 40

ตัวอย่าง (30) = 03

(41) = 36

(42) = 68

(43) = 51

ผลลัพท์ที่ได้ CHECK JUM = 3x2+6+6x2+8+5x2+1

= 43

(40) = 03

จากกรณีเราเขียนผังงานได้ดังรูปที่ 5.19 จะเขียนโปรแกรมได้ดังนี้

```
      LDA  30H     ; COUNT = LENGTH OF STRING (IN BYTE)
      MOV B,A
      MVI C,0      ;CHECKSUM =0
      LXI H, 41 H ; POINT TO START OF STRING
CHDIG MOV A,M      ; GET TWO BCD DIGITS
      MOV D,A      ; SAVE COPY
      RAR
      RAR
      RAR
      DCR B
      JNZ CHDIG
      ANI 00001111B  ; MASK OFF SELF CHECK DIGIT
      HLT
```

การเขียนโปรแกรมที่ยาว ๆ และมีส่วนของโปรแกรมที่ซ้ำ ๆ กันอยู่หลายแห่งอาจยุบส่วนของโปรแกรมย่อยก็คือโปรแกรมที่อาจจะเรียกไปใช้ได้ในขณะที่ซีพียูกำลังประมวลผลในโปรแกรมใดโปรแกรมหนึ่งอยู่ และเมื่อเรียกหาแล้วซีพียูจะกระโดดไปทำโปรแกรมย่อยจนกว่าจะมีคำสั่งหวนกลับ (return) มันจะกลับไปยังโปรแกรมเดิมที่กระโดดมา ลักษณะการทำงานของโปรแกรมย่อยเขียนเป็นไดอะแกรมได้ดังนี้

```
ANI 00001111B       ; MASK OFF MSD
ADD A               ; DOUBLE MSD
DAA                 ; KEEP IT DECIMAL
MOV  C,A
MOV  A,D
ANI 00001111B       ;MASK OFF LSD
ADD C               ; ADD LSD TO CHECKSUM
DAA
MOV C,A
INX  H
```

จากลักษณะการทำงานข้างบนจะเห็นได้ว่าส่วนของโปรแกรมหลัก (main program) ถ้ามีคำสั่งเรียกชื่อโปรกแกรมย่อยมันจะกระโดดไปทำในส่วนโปรแกรมย่อย ครั้นทำในโปรแกรมย่อย จบแล้วจะกลับเข้ามายังโปรแกรมหลักในคำสั่งถัดมาจากคำสั่ง CALL เป็นเช่นนี้

ในการเขียนโปรแกรมย่อยนั้นจะต้องมีเทคนิคที่ต้องคำนึงถึง หลายประการเช่น การคงรักษาข้อมูลบางส่นที่ได้กระทำไว้ก่อนการกระโดดไปทำในส่วนโปรแกรมย่อย เช่น ข้อมูลในรีจิสเตอร์ต่าง ๆ (A,B,C,D,E,H,L) อาจจะต้องเก็บไว้ในที่ใดที่หนึ่งก่อน แล้วก่อนการ RET จึงจะนำเอาข้อมูลเหล่านี้กลับมาใส่ให้ใหม่

กรรมวิธีการเรียกโปรแกรมย่อยเราใช้คำสั่ง CALL ผลที่ได้คือ ซีพียูจะนำข้อมูลที่อยู่ในโปรแกรมย่อยไปให้กับโปรแกรมเคาวน์เตอร์เพื่อให้เครื่องวิ่งต่อไปได้

ลองมาพอจารณาตัวอย่างโปรกแกรมย่อยและโปรแกรมหลักที่ สัมพันธ์กับโปรแกรมย่อยในการนับความยาวของสตริงดังนี้

```
LXI  SP,80      ; START STACK AT LOCATION 80
LHLD 40 H       ; GET STARTING ADDRESS
CALL STLEN      ; DETERMINE STRING LENGTH
STA 42 H        ; STORE STRING LENGTH
```

ส่วนของโปรแกรมย่อยเขียนได้

```
STLEN  MVI    B,0          ; LENGTH = 0
       MVI    A,0DH        ; GET CR FOR COMPARISION
CHKC   CMP M                ; IS CHARACTER 'CR'
       JZ DONE              ; YES END OF STRING
       INR    B             ; NO ADD 1 TO LENGTH
       INX    H
       JMP CHKC
KDONE MOV A,B
       RET
```

ปัญหาที่น่าสนใจอย่างหนึ่งในการใช้โปรแกรมย่อยคือ เรื่องเกี่ยวกับสแตคพอยท์เตอร์ ทั้งนี้เพราะเมื่อเราใช้คำสั่ง CALL ข้อมูลในโปรแกรมเคาวน์เตอร์จะถูกถ่ายไปเก็บไว้ในหน่วยความจำที่สแตคอ้างไว้ และจำนำให้ค่าข้อมูลในสแตคพอยท์เตอร์เองผิดไปจากเดิมนั่นคือการเรียกโปรแกรมย่อยทุกครั้งจะเกี่ยวข้องกับสแตคพอยท์เตอร์

ในส่วนโปรแกรมควบคุมระบบการทำงาน (Monitor) มักเก็บไว้ใน Rom โดยส่วนที่ของสแตคพอยท์จะอยู่ในหน่วยความจำ RAM และการเข้าถึงสแตคเหล่านี้จะต้องเกี่ยวข้องกับแอดเดรสในสแตคมากระทำในส่วนโปรแกรมที่ผู้ใช้เขียนขึ้นส่วนข้องโปรแกรมที่ผู้ใช้เขียนอาจจะมีการเกี่ยวข้องโดยตรงกับสแตคหรือ RAM ในส่วนสแตค ทำให้เมื่อกระโดดกลับเข้ามอนิเตอร์ใหม่จะทำให้การทำงานมอนิเตอร์จึงต้องมีการเก็บข้อมูลในสแตคพอยท์เตอร์ไว้เสียก่อน โดยใช้ส่วนโปรแกรมต่อไปนี้

```
       LXI H,0      ; GET MONITOR  STACK
                      POINTER
       DAD  SP
       SHLD STEMP   ; AND SAVE IT
```

เมื่อเสร็จจากการเก็บข้อมูลในสแตคพอยท์เตอร์ไว้แล้วจึงเรียไปใช้กระทำอย่างอื่นซึ่งอาจจะมีการเซทค่าสแตคพอยท์เตอร์ใหม่อีกครั้งเมื่อต้องการจะกลับเข้าไปในมอนิเตอร์โปรแกรม จำเป็นจะต้องเรียกค่ากลับเข้ามายังสแตคพอยท์เตอร์ใหม่ด้วยชุดคำสั่ง

```
       LHLD  STEMP       ; RESTORE MONITOR STACK POINTER
       SPHL
```

การเขียนโปรแกรมที่เกี่ยวข้องกับตัวอุปกรณ์อินพุทและเอาท์พุท

ก่อนอื่นมาทำความเข้าใจกับโครงสร้างการส่งเสริมการส่งข้อมูลเบื้องต้นระหว่างซีพียูกับอุปกรณ์ I/O อีกครั้ง อุปกรณ์ I/O ที่ซีพียูจะมองเห็นเหมือนแอดเดรสหนึ่ง โดยความเร็วในการส่งข้อมูลก็เหมือนกับการส่งข้อมูลกับ RAM และกากร์ใช้โครงสร้างระดับแรงดันและตัวเลขไบนารี่เหมือนกัน เพียงแต่ว่าการกำหนด I/O ใน 8080 นั้นสามารถกำหนดตัว I/O ได้ถึง 28=256 ตัว และเอาท์พุทถึง 256 คำสั่งว่าซีพียูจะกระทำกับตัว I/O ใน 8080 ก็มีเพียว PORT และ OUT PORT เท่านั้น แต่อย่างไรก็ตามความเร็วในการส่งจากซีพียูไปยังอุปกรณ์เอาท์พุทหรือรับจากอินพุทมายังซีพียูด้วยความเร็วสูงซึ่งขึ้นอยู่กับสัญญาณนาฬิกาที่เราใช้

ในการเชื่อต่อกับอุปกรณ์ I/O นั้นลักษณะของการเชื่อมมักขึ้นอยู่กับความเร็วในการส่งข้อมูลของตัวอุปกรณ์ I/O ด้วยอัตราการส่งข้อมูลออกเป็น 3 ชนิด คือ

1. ชนิดความเร็วช้า ได้แก่ I/O ที่จะส่งข้อมูลหรือส่งข้อมูลมาใช้ด้วยเวลาที่ค่อนข้างช้ามาก เช่น อุปกรณ์จำพวกรีเลย์ และตัวรับความรู้สึกทางกลต่าง ๆ และเราอาจรวมถึงตัวภาคแสดงด้วยแสงบางแบบด้วย

2. ชนิดความเร็วปานกลาง ซึ่งเราอาจกำหนดขนาดความเร็วอยู่ในช่วงประมาณ 1 ถึง 10000 บิทต่อวินาที อุปกรณ์จำพวกคีย์บอร์ดได้แก่ เครื่องพิมพ์ เครื่องอ่านการ์ด เทปคาสเซท อุปกรณ์รับส่งข้อมูลทางอนาลอก เป็นต้น

3. ชนิดความเร็วสูง พวกนี้ส่งข้อมูลได้เร็วกว่า 10000 บิท ต่อวินาที อุปกรณ์พวกนี้ได้แก่ พวกจานแม่เหล็ก เทปแม่เหล็ก เครื่องพิมพ์ความเร็วสูง และ CRT

สำหรับในกรณีของไมโครคอมพิวเตอร์ที่เราให้ความสนใจพิเศษ และเป็นหลักการทางเบื้องต้นนั้น, เราจะสนใจใน 2 กลุ่มนี้ก่อน

ในการรับส่งอุปกรณ์ประเภทมีอัตราการส่งข้อมูลช้า ๆ อาจจะต้องมีการกำหนดเวลาหรือควบคุมสัญญาณจังหวะเวลากันบ้าง เช่น ส่งหรือรับข้อมูลจากเทปคาสเซทด้วยอัตราขนาด 300 บิท ต่อวินาที ดังนั้นแต่ละบิทจะมีเวลาห่างกัน 3.33 มิลลิวินาที ถ้ามากกว่าส่งมา 8 บิท ได้ในคราวเดียวกันก็จะใช้เวลาส่งห่างกัน 8x3.33 = 26.64 มิลลิวินาที ซึ่งก็ใช้เวลานานพอสมควรเมื่อเทียบกับการทำงานของซีพียูที่ทำงานในหน่วยของไมโครวินาที วิธีหนึ่งที่ใช้ในการหน่วงเวลานี้คือ การกำหนดช่วงเวลาด้วยซอฟท์แวร์

เราสามารถเขียนโปรแกรมให้เกิดการดีเลย์ตามที่กำหนดได้ โดยอาศัยผังงานข้างล่างนี้

ค่าของช่วงเวลาที่ดีเลย์ในโปรแกรมนี้ขึ้นอยู่กับการเซท
MSCNT โดยโปรแกรมเขียนได้ดังนี้

```
DELAYMVI B, MSCN ; GET COUNT FOR 1 MS
DLY1 DCR B       ; COUNT =COUNT-1
JNZ DLY1         ; COUNTINUE UNTIL COUNT = 0
DCR A            ; UNMBER OF MS = NUMBER OF MS -1
RET
```

การคำนวณเวลาจากโปรแกรมทำได้ไม่ยากโดยการสร้าง
ตารางเวลาโดยดูว่าในแต่ละคำสั่งใช้เวลาจำนวนกี่ไซเคิลของ
สัญญาณนาฬิกาเราจะได้ตารางคำนวนดังนี้

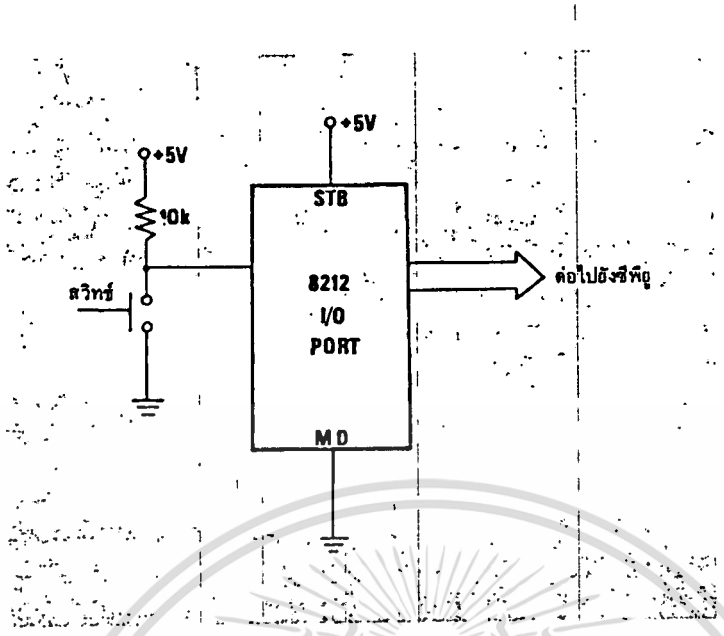
คำสั่ง		หน่วย
MVI B, MSCNT	7x(A)	จำนวนไซเคิลของสัญญาณนาฬิกา
DCR	5x(A)xMSCNT	"
JNZ	10x(A)xMSCNT	"
DCR	5x(A)	
JNZ	10x(A)	"

<u>การอ่านข้อมูลจากสวิทช์</u>

การใช้อุปกรณ์อินพุทแบบพื้นฐานอย่างหนึ่งคือการใช้สวิทช์ซึ่งแต่ละตัวสามารถเป็นอุปกรณ์อินพุทได้ 1 บิท ถ้าเราจะรวมสวิทช์ให้เป็นตัวสามารถเป็นอุปกรณ์ต้องใช้สวิทช์ 8 ตัว ลักษณะของสวิทที่ต่อเป็นตัวอินพุทเมื่ออินเตอร์เฟสเข้ากับซีพียู แบบง่ายหนึ่งแสดงให้เห็นดังรูป 1.10

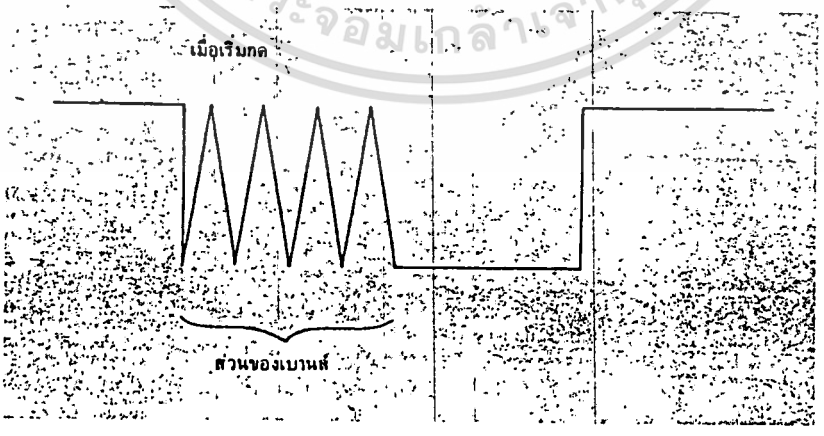
เรามีวิธีการตรวจสอบว่าสวิทช์ได้รับการปิดวงจรหรือยังด้วยคำสั่งง่าย ๆ ที่ขึ้นอยู่กับว่าสวิทช์จะต่ออยู่ที่บิทใดของบัสข้อมูล ในที่นี้สมมติว่าต่อที่บิทที่ 7 คำสั่งที่ใช้คือ

```
IN   PORT   ; READ BUTTON POSSITION
RAL         ; IS BUTTON CLOSED
JNC DONE    ; YES, DONE
```



รูปที่ 1.10

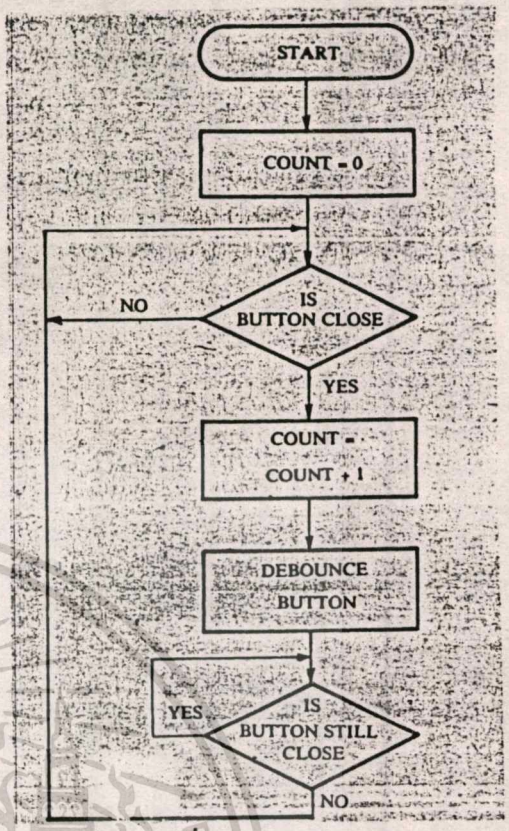
และถ้าเราต่อที่บิท 0 เราก็เพียงเปลี่ยนคำสั่ง RAL เท่านั้น ในการกดสวิทนั้นโดยทั่วไปมักจะต้องเกิดพัลซ์ที่เป็นเบานส์ ลักษณะของเบานส์นี้จะมีผลต่อการอ่านของซีพียูได้ ทั้งนี้เพราะช่วงเวลาการอ่านนั้นถ้าอ่านของซีพียูได้ ข้อมูลที่ได้ก็อาจจะเป็น 0 หรือ 1 ก็ได้ไม่แน่นอน ดังนั้นจึงต้องมีการกำจัดส่วนของเบานส์ การกำจัดส่วนของเบนส์นี้อาจทำได้ทั้งฮาร์ดแวร์และซอฟท์แวร์ แต่วิธีที่จะเป็นที่นิยมมากวิธีหนึ่งการกำจัดเบานส์ด้วย วิธีทางซอฟท์แวร์ เพราะไม่ต้องลงทุนเพิ่มเติมทางฮาร์ดแวร์ลองมาพิจารณาการกำจัดเบานส์จากตัวอย่างนี้



รูปที่ 1.11

การเขียนโปรแกรมจึงต้องคำนึงถึงส่วนเหล่านี้ขอให้พิจารณาผังงาน และเราเขียนโปรแกรมวงจรการนับการกดของสวิทช์ได้ดังนี้

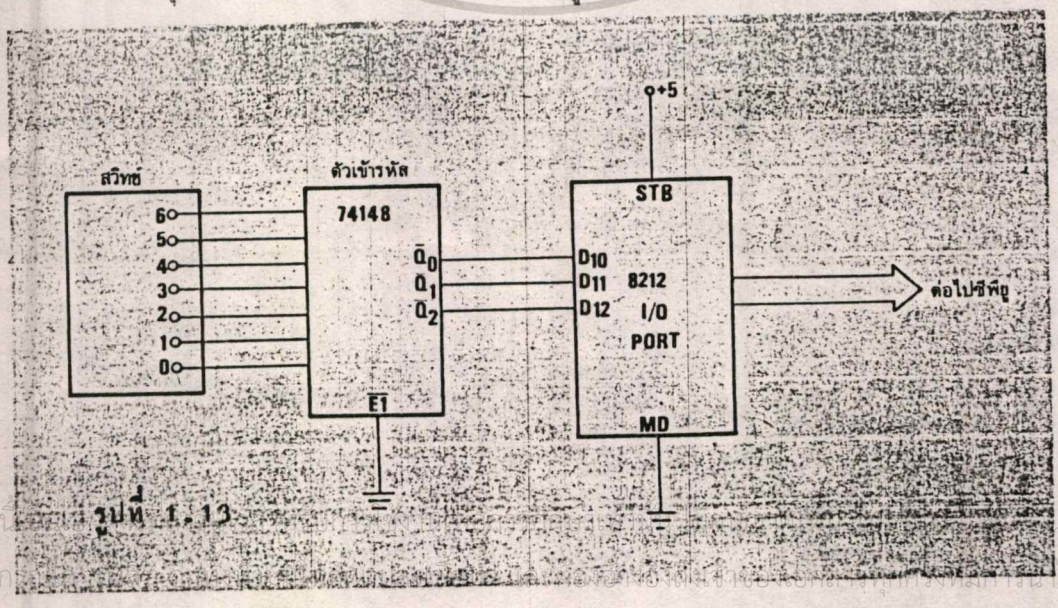
```
        LXI H,40   ; (40) = COUNT =0
        MVI M,0
CHKCL   IN PORT    ; IS BUTTON CLOSED (0)
        ANI MASK
        JNZ CHKCT  ;NO WAIT
        INR M      ; YES,COUNT = COUNT +1
        MVI A,1
        CALLDELAY  ; DEBOUNCE BUTTON BY
                     WAITING IM
CHKOP   -IN  PORT  ; ISBUTTON STILL CLOSED (0)
        ANI MASK   ; YES WAIT
        JZ CHKOP   ; NOLOOK FOR NEXT CLOSERE
        JMP CHKCL
```



รูปที่ 1.12

จากโปรแกรมที่กล่าวมานี้เราจะอ่านการกดของสวิทครั้งแรกแล้วรอเวลาอีกครั้งหนึ่ง ให้ผ่านช่วงเบานส์เสียก่อนแล้วค่อยอ่านอีกครั้ง วิธีการนี้เป็นเทคนิคในการอ่านดีเบานส์เสียก่อนแล้ว ค่อยอ่านอีกครั้ง วิธีการนี้เป็นเทคนิคในการดีเบาส์ ที่ใช้กันทั่วไป (โดยทั่วไปช่วงเวลาดีเลย์จะใช้ประมาณ 10 มิลลิวินาที)

นอกจากนี้เราสามารถดัดแปลงการอ่านสวิทช์เพียงตัวเดียวเหล่านี้มาเป็นลักษณะของสวิทช์เลือกหรือสวิทช์หมุน หรือทัมบ์วิลด์สวิท ด้วยการอ่านรูปลักษณะเช่นวงจรข้างล่าง

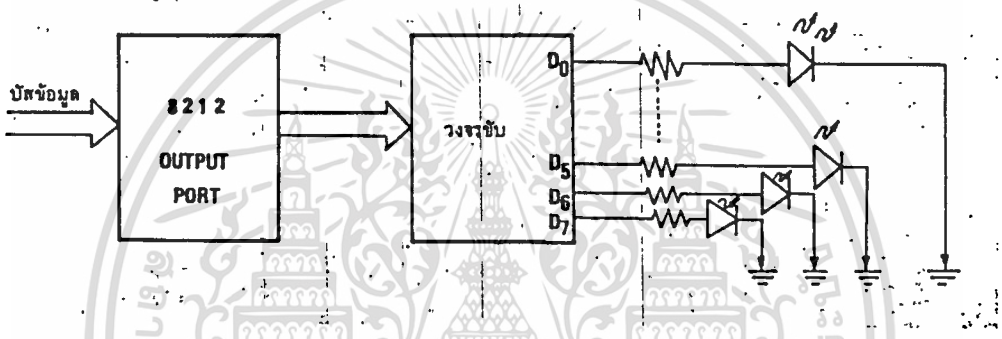


รูปที่ 1.13

การเขียนโปรแกรมอินเตอร์เฟสเข้ากับเอาท์พุทภาคแสดง LED

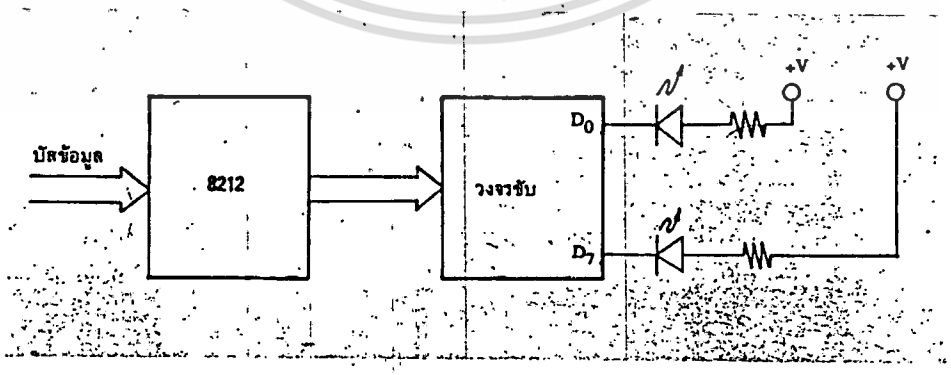
การติดสว่างของ LED จะแสดงสถานะทางลอจิกได้โดยอาศัยการใช้ LED เป็นภาคแสดงผลของไมโครคอมพิวเตอร์ได้อย่างดี การแสดงผลนั้นอาจจะออกมาในรูปของรหัสเลขฐาน 8 หรือฐานสิบหกเลยก็ได้ขึ้นกับวิธีการถอดรหัสภาคแสดงเหล่านั้น

ลักษณะการอินเตอร์เฟสในภาคเอาท์พุท เมื่อแสดงผลลัพท์รหัสไบนารี่จะใช้ไดอะแกรมของวงจรดังตัวอย่างข้างล่างนี้



รูปที่ ก.14

ก. การให้ LED สว่างตามสถานะลอจิก 1



รูปที่ ก.15

ข. การให้ LED สว่างตามสถานะลอจิก 0

การส่งข้อมูลให้ LED สว่างนั้นก็ไม่ยากเย็นอะไรเลย โดยใช้คำสั่งลักษณะนี้คือ

```
MVI A,  MASKP
OUT PORT
```

โดยปกติที่ตัวเอาท์พุทจะมีวงจรในลักษณะที่จะต้องแลทช์ข้อมูลเอาไว้ได้ ดังนั้นเมื่อส่งข้อมูลก็จะสว่างที่ LED ครั้นส่งมาใหม่ ตัว LED ก็จะดับสว่างตามข้อมูลชุดใหม่

การส่งข้อมูลนั้นบางครั้งมาจากข้อมูลในแอดเดรสใดแอดเดรสหนึ่ง ลักษณะการเปลี่ยนข้อมูลโดยโหลดข้อมูลมาจากที่เก็บที่ อยู่ในหน่วยความจำทำได้โดย

```
LXI H, ADDRRE;    :LOAD PONTER TP DATA
MOV  A,M          :GET COPY OF DATA
OUT PORT          :DATA TO LED.
```

จากลักษณะที่กล่าวมานี้เป็นลักษณะของการส่งข้อมูลเข้าแสดงผลด้วย LED ลักษณะตัวเลข 0 และ 1 แต่ถ้าหากต้องเอาท์พุทแสดงผลด้วยตัวแสดง LED 7 ส่วน ที่สามารถใช้ของตัวเลขและถ้าหากไม่ใช่ตำแหน่งจุดทศนิยมก็จะใช้ข้อมูลเนียง 7 บิทเท่านั้น วิธีการที่จะให้ภาคแสดงจะอินเตอร์เฟสในลักษณะที่จะกล่าวนี้ได้วิธีหนึ่ง

จากวิธีการนี้เราจะใช้วิธีการสร้างตารางของภาคแสดงเอาท์พุทดังที่ได้กล่าวมาแล้ว ในที่นี้จะขอกล่าวซ้ำอีกเล็กน้อยในแง่ของการต่อภาคแสดงที่สัมพันธ์กับรหัสในตารางได้ดังนี้

ตัวเลข	เลขฐานสิบหก	
	คาโถด	อาโนด
0	3F	40
1	06	79
2	5B	24
3	4F	30
4	66	19
5	6D	12
6	7D	02
7	07	78
8	7F	00
9	67	18
A	77	08
B	7C	03
C	39	46
D	5E	21
E	79	06
F	71	0E

ลองมาดูตัวอย่างการเขียนโปรแกรมให้แสดงผลต่อภาคแสดงในรูปตัวเลขฐานสิบหก โดยนำข้อมูลในหน่วยความจำมาแสดง ถ้าข้อมูลเก็บตัวเลขเกินกว่า OFH ให้แสดงดับหมด ผังงานจะได้ดังนี้

เราเขียนโปรแกรมตามผังงานได้ดังนี้

```
MVI B,BLANK      ; GET BLANK CODE
LDA ADDR         ; GET DATA
CPI 10 H         ; IS DATA 10 H
JNC  D,SPLY      ; YES, DISPLAY BLANKS
LXI D, SSEG      ; BASE OF 7-SEGMEN TABLE
MVI   H,0
KMOV    L,A      ; MAKE DATA INTO 16 BIT
DAD D            ; INDEX TABLE
MOV B,M          ; GET 7 SEGMENT CODE
OUT PORT         ; CODE TO DDISPLAY
```

จากกรณีดังกล่าวนี้เราจะส่งรหัสไปได้เพียงตัวเดียวแต่ถ้าหากต้องการส่งรหัสไปยังภาคแสดง 7 ส่วนหลาย ๆ ตัวอาจทำได้ในลักษณะการมัลติเพล็กซ์คือส่งไปทีละตัวเรียงกันไปโดยอาจจะถือว่า LED 7 ส่วนแต่ละส่วนคือเอาท์พุท 1 ช่องก็ได้

<u>การเขียนโปรแกรมทำการอ่านคีย์บอร์ดชนิดแมทริกซ์</u>

การใช้งานจริง ๆ ของไมโครคอมพิวเตอร์นั้นลักษณะการอ่านข้อมูลอินพุทจะมีเทคนิคและวิธีการอีกมากมาย เช่น อินพุทที่เป็นคีย์บอร์ดที่มีจำนวนมากมาย ดังตัวอย่างของคีย์บอร์ด ของ CRT หรือ TTY ที่มีจำนวนคีย์ได้ถึงมากกว่า 64 คีย์ การอ่านคีย์จึงมีวิธีการเพื่อให้การอ่านเป็นไปอย่างมีประสิทธิภาพและรวดเร็วมากในที่นี้จะยกตัวอย่างการเขียนโปรแกรมเพื่ออ่านข้อมูลจากคีย์บอร์ดชนิดแมททริกซ์

จะเห็นได้ชัดอย่างหนึ่งว่าถ้าหากจัดทางคีย์บอร์ดแบบขนานเรียงกันตามข้อมูลที่อ่านจะใช้พอร์ทเอาท์พุทมากกว่าหนึ่งพอร์ทในการอ่าน ข้อมูลและยิ่งถ้าจำนวนคีย์บอร์ดยิ่งมากก็ต้องใช้จำนวนพอร์ทอินพุทมากยิ่งขึ้น ลักษณะเช่นนี้อาจลดจำนวนพอร์ทอินพุทลงได้มากด้วยเทคนิควิธีที่เรียกว่าการสแกน

ขนาดของคีย์บอร์ด	จำนวนสายถ่ายวาง ขนานกัน	จำนวนสายถ่ายวาง แบบเมททริกซ์
3x3	9	6
4x4	16	8
4x6	24	10
5x5	25	10
6x6	36	12
6x8	48	14
8x8	64	16

สิ่งที่นักโปรแกรมจะต้องทำการค้นหาว่ามีคีย์ใดได้รับการกดลงมา การค้นหาทำได้ด้วยการสแกนทีละแถว เริ่มที่แถว 0 ก่อนแล้วตรวจแต่ละคอลัมน์ดูถ้าไม่มีก็เลื่อนมาที่แถวที่ 1 แล้วตรวจแต่ละคอลัมน์ ทำเช่นนี้จนครบทุกแถวก็จะพบว่ามีคีย์ใดได้รับการกดถ้าหากว่าไม่มีการกดในคีย์ใดเลยก็ให้วนตรวจสอบใหม่ ขอให้พิจารณารูป 1.16

จากกรณีนี้เราจะส่งสัญญาณจากเอาท์พุทแลทช์ไว้ก่อน แล้วตรวจสอบด้านการอ่านข้อมูลทางด้านอินพุทว่ามีคีย์ใดที่ลัดวงจรต่อถึงกัน ในกรณีนี้สมมติให้ซีพียูส่งสัญญาณมากกราวนด์ทางด้านแถวไว้โดยการสแกนทีละแถวแล้วอ่านข้อมูลทางด้านอินพุทพอร์ท

ลองดูตัวอย่างการอ่านว่าการกดคีย์เกิดขึ้น เราจะใช้วิธีการตรวจสอบได้ดังนี้

```
    OUT KBDOT       ; GROUND ALL KEYBOARD ROWS
    IN  KBIN        ; GET KEYBOARD COLUMN DATA
    ANI 00000111B ; MASK COLUMN BITS
    CPI 00000111B ; ARE ANY COLUMNS GROUND
    JZ  WAITK       ; NO KEEP LOOKING AT KEY BOARD
DONE JMP      DONE
```

จากโปรแกรมจะเห็นว่าเราส่งข้อมูลไปยังแต่ละแถวด้วยลอจิก "0" ก่อนแล้วอ่านข้อมูลมาทางอินพุทตรวจสอบดูว่ามีคอลัมน์ใดอ่านได้ลอจิก "0" หรือไม่อ่านก็แสดงว่ามีการคีย์ใดคีย์หนึ่ง

การวางรูปและการเขียนโปรแกรมสำหรับมอนิเตอร์

โดยปกติการทำงานในระบบของคอมพิวเตอร์จะต้องอยู่ภายใต้การควบคุมของโปรแกรม ถ้าหากว่าเป็นเครื่องคอมพิวเตอร์ขนาดใหญ่ โปรแกรมที่ใช้ควบคุมระบบนี้เรียกว่า OS (OPERATING SYSTER) สำหรับไมโครคอมพิวเตอร์ขนาดเล็ก เช่น ไมโครคอมพิวเตอร์แผงเดียว โปรแกรมควบคุมการจัดระบบจะเรียกว่า มอนิเตอร์โปรแกรม

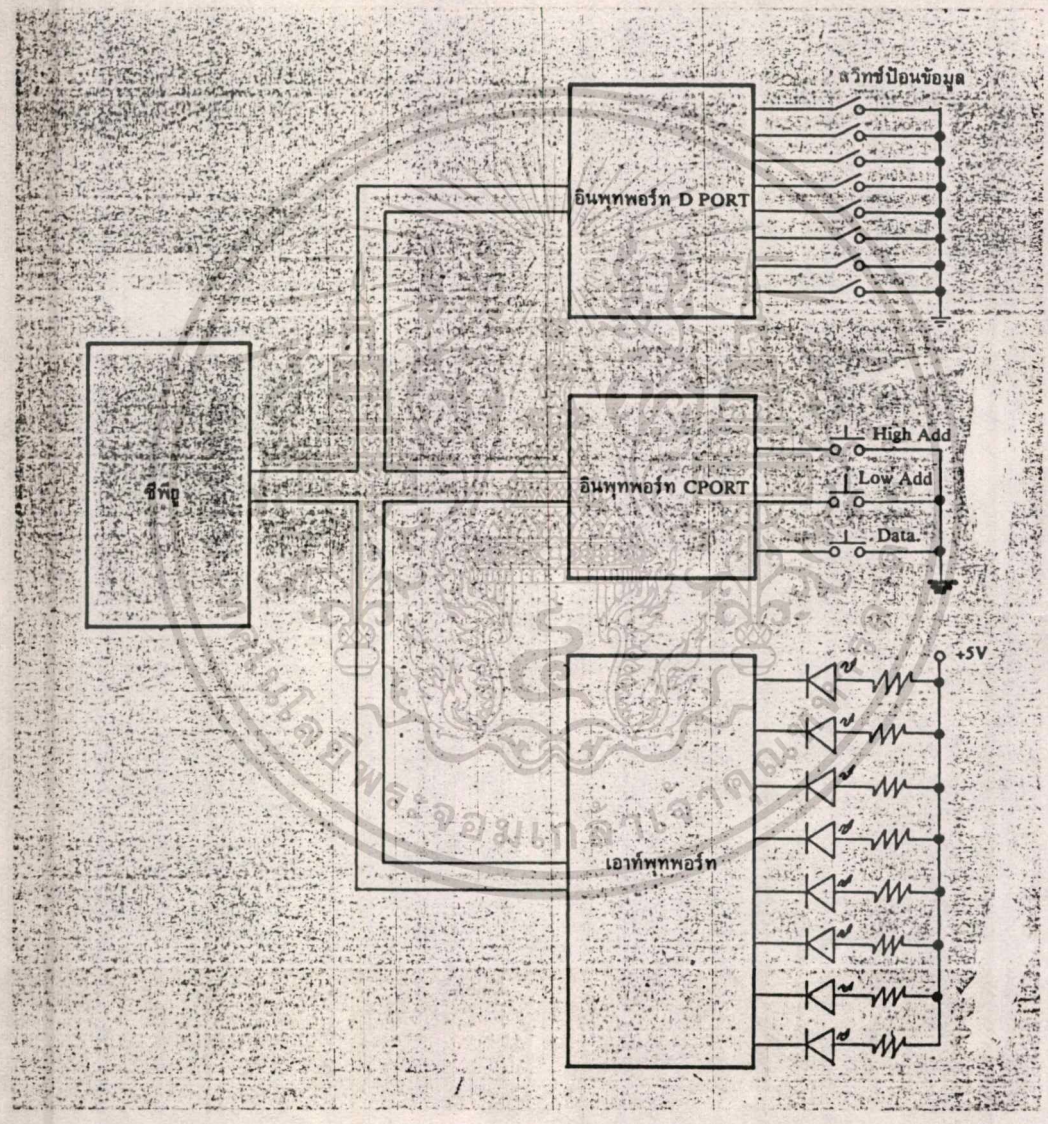
โดยปกติขีดความสามารถของไมโครสามารถของไมโครคอมพิวเตอร์ขนาดเล็กจะต้องประกอบด้วยทั้งฮาร์ดแวและซอฟท์แวร์ที่ดี สำหรับซอฟท์แวร์นั้นก็ขึ้นอยู่กับโปรแกรมมอนิเตอร์ที่ผู้เขียนจะเพิ่มฟังชั่นหรือขีดความสามารถต่าง ได้เพียงไร

เมื่อเราเปิดสวิทช์ป้อนไฟเลี้ยงให้กับเครื่องไมโครคอมพิวเตอร์จะเห็นว่าตัววงจรจะทำการรีเซทตัวเองในขณะเปิดไฟ หรือถ้าไม่รีเซทตัวเองก็มักจะต้องมีสวิทช์เพื่อทำการรีเซทก่อนการใช้งาน การรีเซทตัวเองมักจะต้องมีสวิทช์เพื่อทำการรีเซทก่อนการใช้งาน การรีเซทโดยทั่วไปก็คือการทำให้ค่าใน PC มีค่าเป็น 0 ก่อน แล้วให้เครื่องคอมพิวเตอร์เริ่มทำตามโปรแกรมตั้งแต่แอดเดรส 0 ดังนั้นที่ตำแหน่งแอดเดรส 0 จึงต้องมีคำสั่งส่งให้เครื่องได้ทำงานคำสั่งในตอนแรกนี้อาจจะเป็นคำสั่งที่ทำให้คอมพิวเตอร์กระโดดไปที่แอดเดรสอื่นๆ ได้อีกมากมาย ครั้นเมื่อเครื่องเริ่มรันโปรแกรมมันจะเข้าสู่โปรแกรมที่เรียกว่า มอนิเตอร์

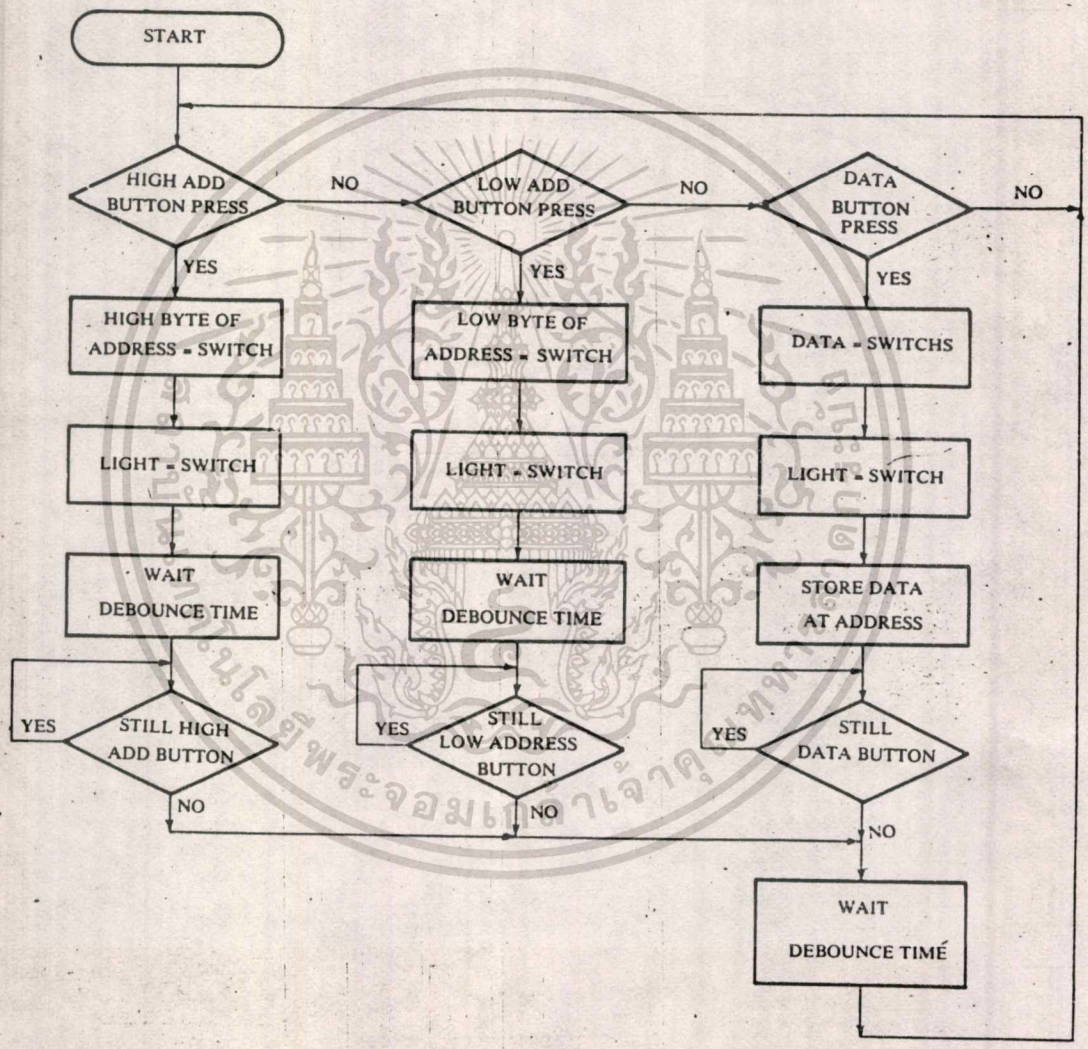
ลักษณะของโปรแกรมมอนิเตอร์ก็มักจะจัดให้เครื่องทำงานเช่น จัดให้อ่านข้อมูลที่คีย์มาจากคีย์บอร์ด เมื่ออ่านข้อมูลแล้วก็จะตีความหมายของคีย์บอร์ดนั้นว่าเป็นแอดเดรส เป็นข้อมูลโดยโปรแกรมมอนิเตอร์จะเอาส่วนของข้อมูลนั้นมาตีความหมายและกระทำตามอีกครั้ง

เพื่อให้เข้าใจหลักเบื้องต้นของมอนิเตอร์ จึงขอยกตัวอย่างของวงจรพื้นฐานเบื้องต้นอย่างง่ายๆ ของไมโครโปรเซสเซอร์ในการใช้ทำเป็นเครื่องไมโครคอมพิวเตอร์แผงเดียว

รูปที่ 1.16



ระบบนี้เป็นระบบง่ายๆ ที่ใช้ป้อนข้อมูลจากสวิทช์ไปเก็บยังหน่วยความจำใน 1 ตำแหน่งที่ต้องการ โดยที่เราสามารถเซทค่าสวิทช์ที่ส่วนของสวิทช์ป้อนข้อมูลได้ทั้งเป็นค่าแอดเดรสหรือข้อมูลซึ่งขึ้นอยู่กับกดสวิทช์คำสั่งควบคุม ผังงานของมอนิเตอร์ที่จะใช้เป็นตัวป้อนข้อมูลได้ดังนี้

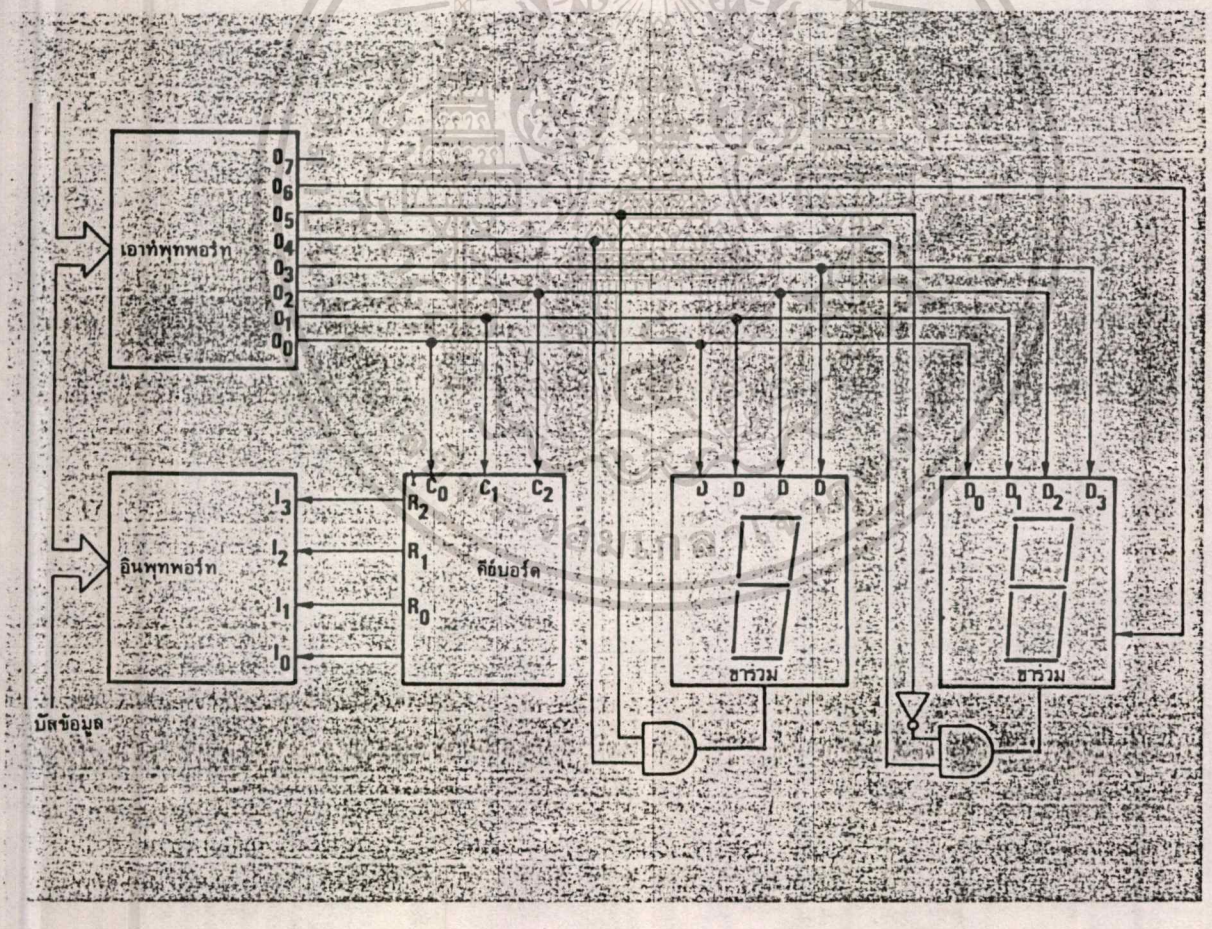


รูปที่ 1.17

ตัวอย่างการใช้โปรแกรมการประยุกต์แทนวงจรทางฮาร์ดแวร์

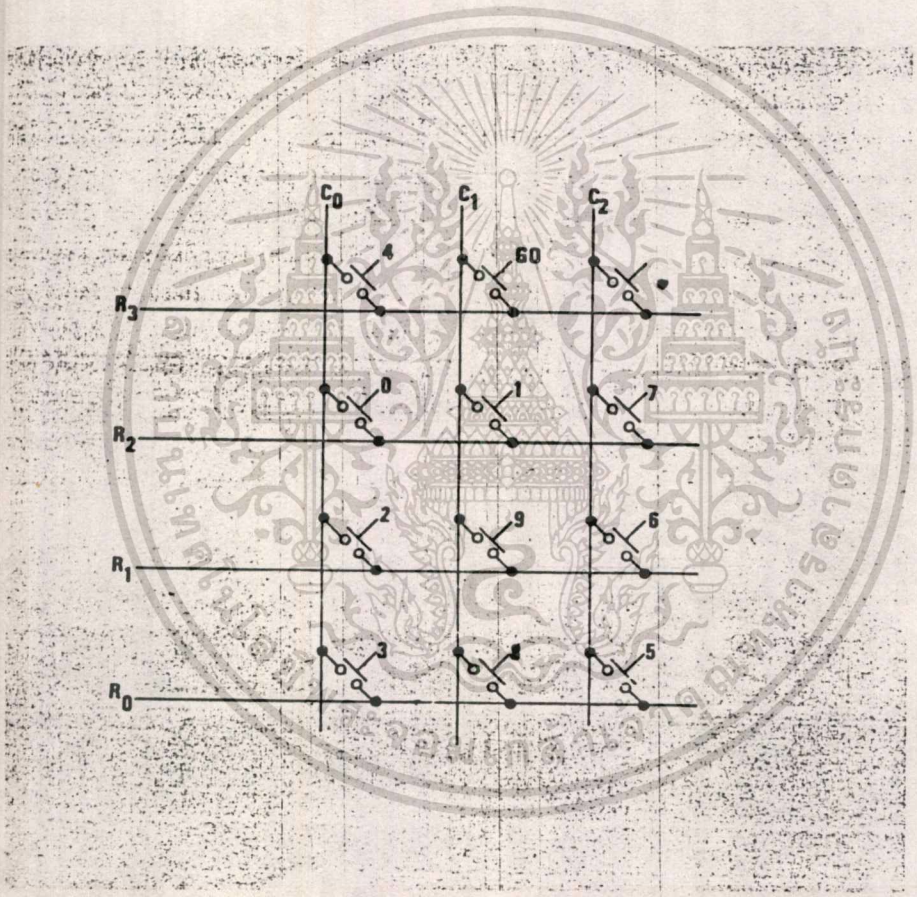
ลักษณะของตัวอย่างที่จะยกขึ้นมากล่าวในที่นี้เป็นลักษณะของนาฬิกาจับเวลาที่เราสามารถโปรแกรมตัวเลขได้ ๒ ตัวเลข ซึ่งเป็นหลักนาทีและหลักสิบของนาที การโปรแกรมนั้นเราจะโปรแกรมผ่านทางคีย์บอร์ดเมื่อป้อนตัวเลขครบสองตัวแล้วก็จะกดคีย์ GO วงจรจะแสดงการนับแบบนับถอยหลังโดยภาคแสดงที่แสดงผลจะค่อย ๆ ลดค่าลงที่หนึ่งจนมาเป็น 0 แล้วจะหยุด

ในการที่จะโปรแกรมการทำงานให้กับวงจรนั้นเราต้องมาศึกษาและวางรูปโปรแกรมให้กับเครื่องเสียก่อนในกรณีนี้ถ้าเราให้การคีย์ตัวเลขเครื่องก็จะนำตัวเลขที่เราคีย์ ไปบนแผงแสดงครั้นเมื่อตัวเลขปรากฏครบสองตัวแล้ว เราคีย์ GO วงจรก็จะเริ่มการเป็นนาฬิกาจับเวลาทันที โดยจะค่อยๆ ลดค่าตัวเลขตามช่วงเวลาที่ตัวไว้จนกระทั่งถึง 0 ผังงานของวงจรเขียนได้ดังนี้



รูปที่ 1.18

เพื่อให้การอ่านดปรแกรมที่เขียนไว้ได้ง่ายเข้าในที่นี้ขออธิบายล้าดับขั้นของโปรแกรมพร้อมเทียบกับผังงานที่กล่าวแล้ว โดยโปรแกรมที่เขียนขึ้นในแบบฉบับมาตรฐานของภาษาแอสเซมเบลอร์



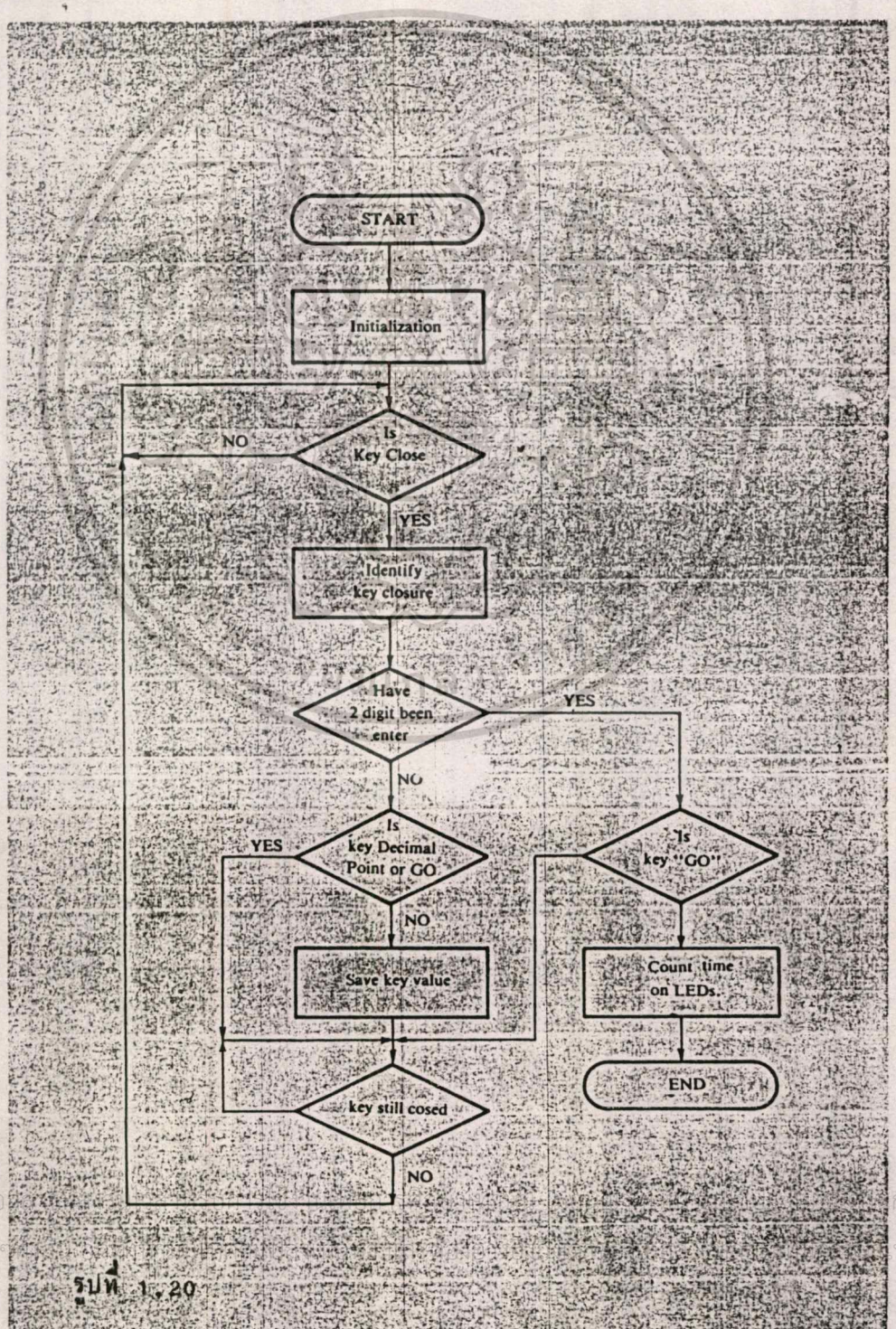
รูปที่ 1.19

สำหรับโปรแกรมที่เขียนขึ้นนั้นไม่จำเป็นต้องเริ่มที่ 0 จึงต้องมีคำสั่งกระโดดไปที่ตำแหน่งที่เริ่มต้นของโปรแกรม โดยปกติการกำหนดค่าเริ่มต้นจะประกอบด้วยลำดับขั้นใหญ่ ๆ 3 ขั้น

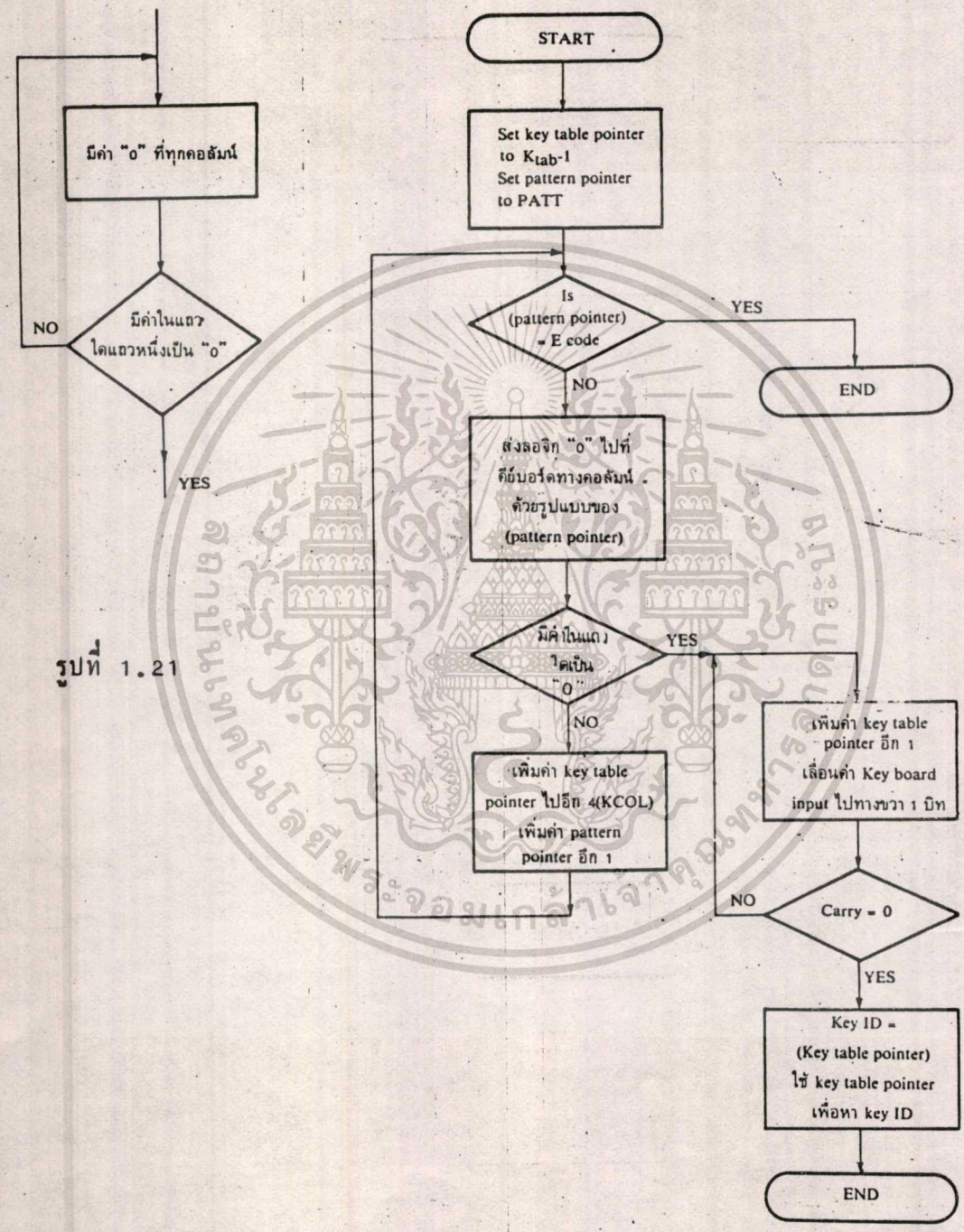
ก. กำหนดแอดเดรสให้กับสแตคว่าจะให้อยู่ที่แอดเดรสใดโดยการโหลดค่าข้อมูลแอดเดรสมาเก็บไว้ที่สแตคพอยท์เตอร์

ข. กำหนดค่าเริ่มต้นที่จำเป็นเช่นค่าที่ใช้ในการตรวจสอบที่กดคือค่า numbdr of digit keys pressed (NKEY)ให้เป็น 0

ค. กำหนดค่าของ KEYAD ซึ่งค่านี้จะเป็นตัวกำหนดแอดเดรสใน RAM และที่ตำแหน่งแอดเดรส KEYAD นี้จะเก็บค่าตัวเลขที่จะได้รับการคีย์ถัดไป และเมื่อโปรแกรมรับค่าที่คีย์มันจะเพิ่มค่า KEYADขึ้นไปอีกหนึ่งทุกครั้ง



รูปที่ 1.20



รูปที่ 1.21

รูปที่ 1.22

บทที่ 2

ระบบหน่วยความจำ (Memory Systems)

ตามปกติ ไมโครโปรเซสเซอร์เป็นหัวใจของระบบไมโครคอมพิวเตอร์ทั่วๆ ไป และหน่วยความจำเปรียบเสมือนสมองของระบบ ซึ่งมีความสำคัญไม่ยิ่งหย่อนไปกว่าตัวไมโครโปรเซสเซอร์เอง ที่จริงแล้วหน่วยความจำยังเป็นส่วนที่มีราคาแพงมากในระบบอีกด้วย ดังนั้นจึงควรได้รับความสนใจอย่างจริงจัง หน้าที่สำคัญของหน่วยความาจำในระบบไมโครคอมพิวเตอร์ก็คือ

1. เป็นหน่วยเก็บโปรแกรมคำสั่งที่จะสั่งให้ CPU ทำงานตามความต้องการ

2. เป็นหน่วยเก็บข้อมูลชั่วคราว ซึ่งหมายถึงข้อมูลสามารถเก็บเข้าไปในหน่วยความจำหรือเอาออกจากหน่วยความจำได้ตามความต้องการ ในขณะใช้งานหรือไม่ใช้ก็ได้

จะเห็นจากกาหน้าที่ทั้งสองของหน่วยความจำว่า ในบางครั้งส่งที่ถูกเก็บไว้อาจเป็นในลักษณะถาวร นั่นคือไม่ต้องเปลี่ยนแปลงอีกต่อไป หลังจากเก็บไว้แล้ว เช่นเป็นโปรแกรมสั่งงานของ CPU ในงานประจำ แต่ในบางครั้ง อาจมีการเก็บข้อมูลในลักษณะชั่วคราวเช่นเป็นตัวเลขจากการคำนวณพร้อมจำนำไปใช้งานต่อไป จากความจำเป็นนี้ ทำให้ต้องมีหน่วยความจำมากกว่า 1 ประเภท เช่น อาจเป็นหน่วยความจำถาวร และหน่วยความจำชั่วคราวที่สามารถเก็บและเรียกออกมาใช้งานได้ทุกเมื่อ สำหรับในภาคนี้จะแบ่งเป็น 2 ตระกูล คือ Rom กับ Ram

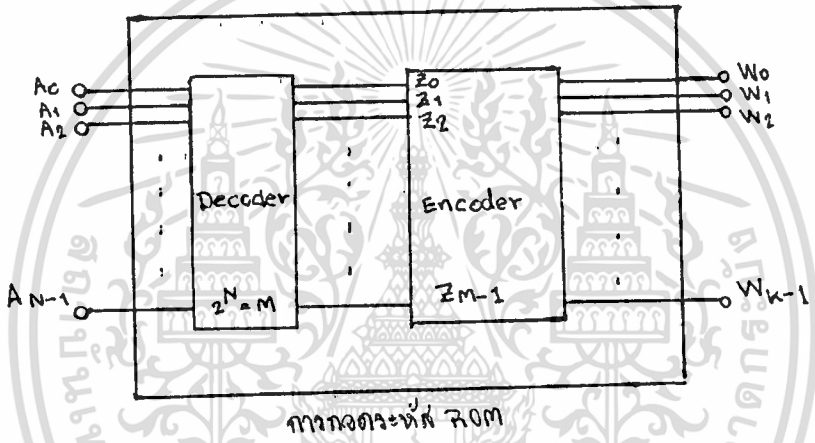
คุณสมบัติที่น่าสนใจ 2 อย่างของหน่วยความจำ ได้แก่ DESTRUCTIVENES คือการที่เราอ่านข้อมูลออกมาแล้ว หน่วยความจำยังจำข้อมูลนั้นได้หรือไม่ และ VOLATILITY คือการที่เมื่อแหล่งจ่ายไฟยังจำข้อมูลนั้นได้หรือไม่ และหน่วยความจำในสมัยแรก ๆ ซึ่งเป็นพวก Magnetic Core Memory มักเป็น Desturctive Memory แต่ Volatility ขึ้นอยู่กับชนิดและตระกูลของหน่วยความจำนั้น เช่น ROM มักเป็นพวก Nonvolatile ส่วน RAM มักเป็นพวก Volatile จึงาตอ้งได้รับการเอาใจใส่เป็นพิเศษโดยเฉพาะในแง่ของแหล่งจ่ายไฟเลี้ยงวงจร

ตระกูล ROM

ROM (Read-Only Memory) เป็นหน่วยความจำที่ใช้เก็บข้อมูลแบบถาวร หรือกึ่งถาวร นั่นคือข้อมูลต้องถูกเขียนไว้ในหน่วยความจำตั้งแต่ต้น หลังจากนั้นก็ปีนการอ่านข้อมูลออกมาเท่านั้น การเขียนข้อมูลทำได้ 2 วิธีคือ การเขียนข้อมูลจากโรงงานผู้ผลิต ตามความต้องการของลูกค้า เช่นเป็นชนิด Mask-Programmed ส่วนอีกแบบหนึ่ง คือผู้ใช้สามารถโปรแกรมได้เองเรียกว่าเป็นชนิด PROM (Programmable Read-Only Memory) และก็มีบางประเภทที่ผู้ใช้สามารถเปลี่ยนข้อมูลจากหน่วยความจำมาก ประเภทนี้เป็นชนิด Reprogrammable ROM หรือบางาครั้งอาจเรียกว่า Read-Mostly Memory (RMM)

ข้อดีอย่างหนึ่งของหน่วยความจำตระกูลนี้ คือข้อมูลจะไม่สูญหาย ถึงแม้ไฟฟ้าจะดับก็ตาม เราเรียกลักษณะอย่างนี้ว่า Nonvolatile ซึ่งต่างไปจากหน่วยความจำตระกูล RAM ซึ่งเป็นชนิด Volatile

ขอให้ดูรูปที่ 2.1 ซึ่งแสดงให้เห็นโครงสร้างภายในของ Rom ทั่วไป ตามรูป A0 ถึง AN-1 เป็นแอดเดรสของข้อมูบที่ต้องการอ่านซึ่งหลังจากผ่านวงจรถอดรหัส จะมีเพียงสายเดียวจากจำนวนทั้งหมด 2N= M สายที่ถูกกระตุ้น และหลังจากผ่านวงจรเข้ารหัสแล้ว ข้อมูลจะมาปรากฏที่ขั้วออก W0 ถึง WK-1 จะเห็นว่าสัญลักษณ์เข้าที่มี N บิท และทำหน้าที่เสมือนเป็นแอดเดรสจะถูกแปลงเป็นสัญญาณออก K บิท ซึ่งก็คือ ข้อมูบที่เราต้องการโดยไม่สนใจว่า N จะมากกว่าหรือเท่ากับหรือน้อยกว่า K ในที่นี้จะบอกได้ว่าหน่วยความจำตระกูล ROM นี้ มีขนาดเป็น MxK บิท ที่ M คือ จำนวนเวิร์ด ทั้งหมดในหน่วยความจำ และ K คือจำนวน บิทของแต่ละเวิร์ด์



รูป ที่ 2.1 โครงสร้างภายในของ ROM

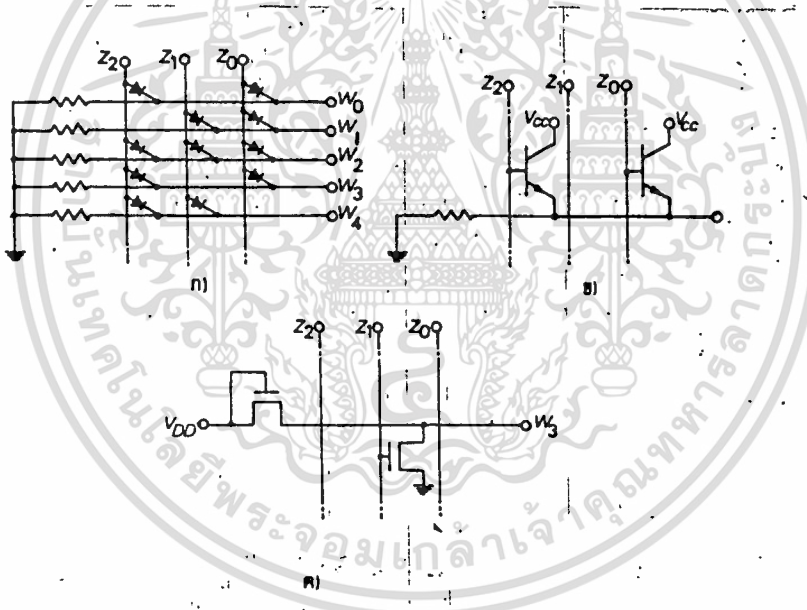
ชนิดของ ROM

- Mask-Programmed ROM

Mask-Programed ROM เป็นหนวยความจำที่โปรแกรมไว้อย่างถาวรในระหว่างขบวนการผลิต โดยการทำ Mask บางอันเฉพาะสำหรับงานนั้น ๆ ROM ประเภทนี้สามารถทำได้จากสิ่งประดิษฐ์จำนวก ไดโอด ทรานซิสเตอร์ หรือ MOSEET อีกทั้งมีการสูญเสียกำลังค่อนข้างสูง ROM ที่ทำจาก BJT จึงมีจำนวนบิทน้อยกว่า ROM ที่ทำจาก MOSFET บนชิ้นสารกึ่งตัวนำที่มีพื้นที่เท่ากันของหน่วยความจำ จะเห็นจากรูปที่ 2.2 ซึ่งแสดงเฉพาะภาควงจรเข้ารหัสของ ROM ว่า สายแอดเดรสและสายสัญญาณออกทั้งหมดจะประกอบกันเป็นตารางกริด ที่จุดตัด จะมีสิ่งประดิษฐ์อาจเป็นไดโอดหรือ BJT หรือ MOSFET ก็ได้ ต่อเชื่อมโยงระหว่างสายแอดเดรสกับสายสัญญาณออกหรือไม่นั้น ขึ้นอยู่กับว่า ที่บิทของแอดเดรสนั้น ๆ ต้องการให้เป็น 1 หรือ 0

- Programmable ROM (PROM)

ที่จริงแล้ว PROM นั้น มีลักษณะวงจรภายในคล้ายคลึงกับพวก Mask-Programmed ROM และตามปกติมักจะทำเป็นตัว IC ที่มีขาอย่างเดียวกับ Mask-Programmed ROM ที่เป็นคู่ของมันด้วย ทางผู้ผลิตได้ต่อหรือไม่ต่อเชื่อมโยงสายแอดเดรสกับสายสัญญาณออก ด้วยสิ่งประดิษฐ์สารกึ่งตัวนำที่จุดตัวในตารางกริดตามที่เราต้องการให้เป็น 1 หรือ 0 ไว้เรียบร้อย ส่วนใน PROM ที่จุดตัดจากจุดในตารางกริด จะมีสิ่งประดิษฐ์เผื่อให้ฟิวส์หลอมขาด ตัดการต่อเชื่อมโดยยิงที่บิทนั้น ๆ หลังจากโปรแกรมเสร็จก็จะเหลือเฉพาะบิทที่ต้องการให้มีการต่อเชื่อมโยงด้วยสิ่งประดิษฐ์ลักษณะอย่างนี้ทำให้มีอีกชื่อหนึ่งว่า Field-Programmable ROM ได้อย่างดีและโปรแกรมได้ง่ายด้วย

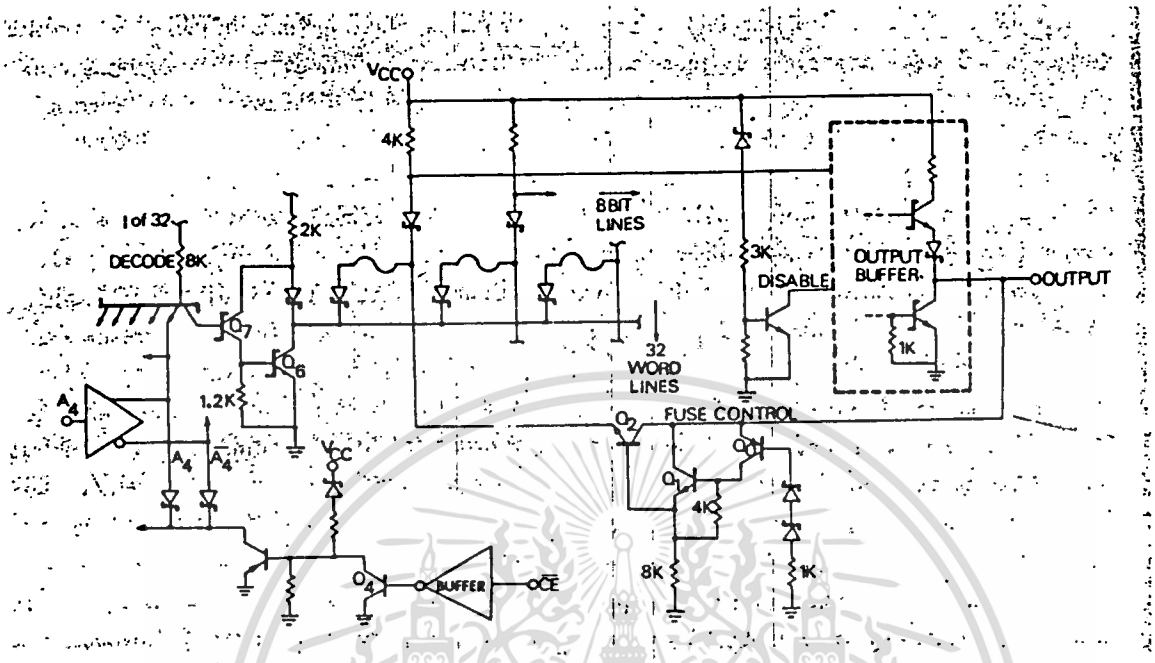


รูปที่ 2.2 วงจรภายในของภาควงจรเข้ารหัสของ ROM

ก. ไดโอด

ข. ทรานซิสเตอร์

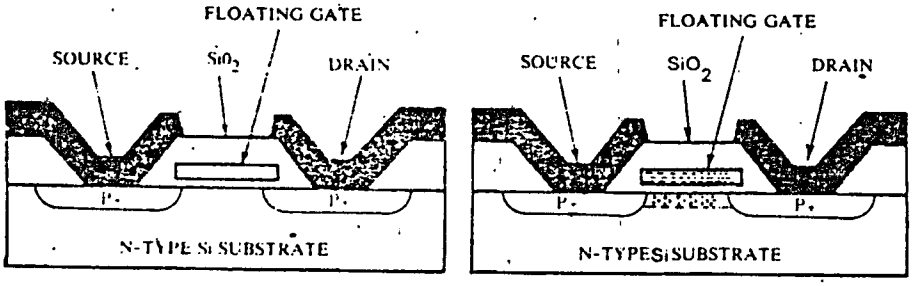
ค. MOSFET



รูปที่ 2.3 วงจรภายในบางส่วนของ PROM

ลักษณะโครงสร้างภายในและแนวความคิดของROMชนิดนี้ แตกต่างจากของPROM มาก ตามปกติ Reprogrammable ROM ทำจาก MOSFETโดยที่สภาวะ 0 หรือ 1 แทนการนำกระแสหรือไม่นำกระแสของ MOSFET ซึ่งการนำกระแสหรือไม่ของ MOSFET จะเห็นว่า ขาเกทถูกแยกจากส่วนที่นำกระแสอย่างเด็ดขาด ดูรูป 2.4 ดังนั้น แรงดันที่ขาเกทจึงขึ้นกับประจุที่ขาเกท ลักษณะโครงสร้างของขาเกทกับส่วนนำกระแส เปรียบเสมือนตัวเก็บประจุ ที่มีชั้นออกไซด์เป็นตัวกลางไดอิเลคตริค นี่เองเป็นแนวความคิดในการพัฒนา Reprogrammable ROM ขึ้นมา โดยวิธีการดักประจำไว้ที่ขาเกท และประจุนั้นสามารถค้างอยู่ที่ขาเกทได้เป็นเวลานาน

แนวความคิดดังกล่าวยังแยกออกได้เป็น 2 วิธี ทั้งนี้ขึ้นอยู่กับผลของการดักประจุในขาเกท วิธีแรกคือการดักประจุไว้ที่ขาเกทซึ่งถูกแยกจากส่วนนำกระแสของ MOSFET ด้วยชั้นของ ซิลิกอนไดออกไซด์ ทำให้ MOSFET ทำให้พาหะประจะข้อมูลที่ถูกโปรแกรมไปแล้ว ทำได้วิธีเดียวคือการลบ โดยฉายด้วยรังสีอุลตราไวโอเล็ตเพื่อให้พาหะประจุที่ถูกดักอยู่ มีพลังงานพอเพียง



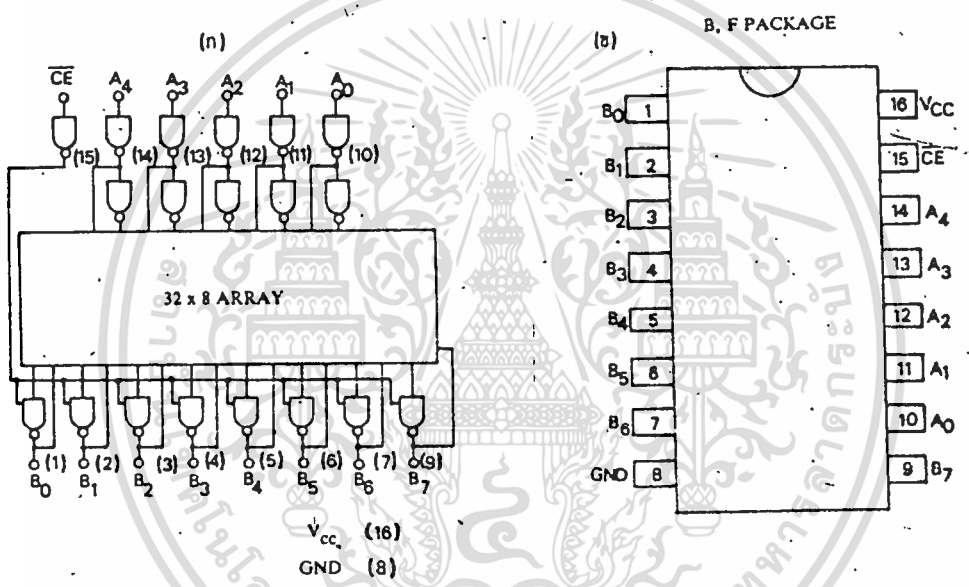
รูปที่ 2.4 โครงสร้างของ MOSFET

ตัวอย่างและการโปรแกรม PROM และ EPROM

ในที่นี้จะยกตัวอย่างเพียงชนิด PROM และ EPROM เพราะเป็นหน่วยความจำที่เราสามารถใช้งานได้กว้างกว่า ROM ชนิดอื่น อีกทั้งเป็นพื้นฐานในการเข้าใจวิธีการโปรแกรม ROM ทั้ง 2 ประเภทในหัวข้อถัดไป

ประเภทที่จะพูดถึงคือ PROM ขอยกตัวอย่าง IC เบอร์ 8s 23 /123 ซึ่งเป็น Bipolar PROM ขนาด 32x8 บิท (ที่จริงแล้วเป็นชนิด Schottky-Diode Clamped Transistor)

การทำงานอย่างคร่าว ๆ เมื่อเลือกแอดเดรส (A0 -A4) แล้ว แอดเดรสจะถูกลงรหัสเป็น 1 ใน32 สาย ซึ่งาจะกระตุ้นเซลหน่วยจำทั้ง 8 บิท ผ่าน Q7 และ Q6 ในขณะเดียวกัน เราต้องส่งสัญญาณมาด้วยว่าเรากำลังเลือกใช้ IC ตัวนี้ โดยบังคับให้ขา CE เป็น 0 หลังจากผ่านบัฟเฟอร์และ Q4, มันจะบังคับให้เซลทั้ง 8 บิท ของหนวยความจำแอดเดรสนี้ทำงาน แรงดันที่ขั้วออกแต่ละบิท (B0 - B7) ในรูป 2.5 ที่ต้องการโปรแกรม(หลังจากโปรแกรมแล้ว บิทนั้นจะเป็น 0) จะต้องถูกยกระดับขึ้นให้สูงพอ ทำให้ Q0, Q1, Q2 น้ากระแสได้ ในขณะที่ Q6, Q7 ก็น้ากระแสด้วยเช่นกัน จึงมีกระแสไหลผ่านฟิวส์ได้มากพอทำให้ดิวส์หลอมขาด สวนบิทใดที่ไม่ต้องการโปรแกรม ก็ไม่ต้องให้แรงดันที่ขั้วออกนั้น ซึ่งไม่มีกระแสไหลผานฟิวส์ บิทนั้นยังคงเป็น 0 และขั้วออกของแต่ละบิทยังผ่านบัฟเฟอร์ที่สามารถเอนเอเบิลได้ตามความต้องการ โดยบังคับด้วยสาย CE



รูปที่ 2.5 ก. รูปตรรกของ 82 S 23/123
ข. รูปแสดงชื่อสัญญาณที่ขาต่าง ๆ

รูปที่ 2.6 แสดงวงจร Manual Programmer เครื่องโปรแกรมโดยใช้มือของ 82 S 2 /123 รวมทั้งแสดงรูปแสดงลำดับตาง ๆ ที่เกี่ยวข้องกับการโปรแกรม PROM วงจรนี้สามารถดัดแปลงเป็นเครื่องโปรแกรมอัตโนมัติได้ไม่ยากนัก (แต่ยังไม่มีความจำเป็นในรูปขณะนี้) R1 เป็นวงจรควบคุมแรงดันคงที่ 5 โวลต์ (Vcc ในรูป) สำหรับตรวจสอบสภาวะ (0 หรือ 1) ของเซลหนวยความจำที่ถูกที่ต้องการโปรแกรม R2 เป็นวงจรควบคุมกระแสคงที่ เนื่องเป็นแหล่งจ่ายกระแสไฟฟ้าให้ฟิวส์หลอมอย่างตามต้องการ

สัญญาณสตาร์ทดังรูปแสดงลำดับช่วงเวลาในการโปรแกรม หลังจากนี้ T1 - T5จะให้สัญญาณพัลซ์

ตามรูปแสดงลำดับช่วงเวลาในการโปรแกรมดังนี้

1. T1 จะเป็น 1 เป็นเวลา 5 มิลลิวินาที ระว่างนี้ Q1 ถูกบังคับให้หยุดนำกระแสส่วน Q2 ยังคงไม่นำกระแส ดังนั้น สาย CE ยังเป็น 1 อยู่ PROM จึงยังไม่ถูกกระตุ้นให้ทำงานวงจรขับ 1 ของ 7005451 จะบังคับให้ R2 จ่ายแรงดัน 10 โวลต์ให้ PROM

2. T2 ซึ่งเป็น 1 พร้อมกับ T1 โดยเป็น 1 เป็นเวลาเพียง 1 มิลลิวินาที เป็นเพียงสัญญาณชะลอเท่านั้น

3. T3 จะเป็น 1 ในทันทีที่ T2 เปลี่ยนเป็น 0 และ T3 เป็นเวลาเนียง 3 มิลลิวินาที เพื่อเปิดทางให้กระแสจากแหล่งควบคุมกระแส R3 พร้อมที่จะจ่ายกระแสให้แก่บิทที่ต้องการโปรแกรมผ่านทางวงจรขับ 2 ของ 75451

4. T4 ซึ่งเป็น 1 พร้อมกับ T3 โดยเป็น 1 เป็นเวลาเพียง 1 มิลลิวินาที เป็นเพียงสัญญาณชะลอเท่านั้น

5. T5 เป็นสัญญาณเอเบิล PROM โดยบังคับให้ Q2 นำกระแสเป็นเวลา 1.5มิลลิวินาที ทำให้สาย CE เป็น 0 ในช่วงเวลานี้ เซลหน่วยความจำในเฉพาะแอดเดรสที่ถูกกำหนดโดยสวิท

รูปที่ 2.7 แสดงรายละเอียดสเปคในการโปรแกรม PROM 82S 23/123 ซึ่งเป็นเวลาที่มาของวงจรเครื่องโปรแกรมโดยใช้มือ และรูปแสดงลำดับช่วงเวลาในการโปรแกรมในรูปที่ 2.6 ข้อที่ต้องระวังในการโปรแกรม PROM ตัวนี้ เพื่อให้ได้ผลที่น่าเชื่อถือ ก็ได้แก่สิ่งต่อไปนี้

1. แหล่งควบคุมแรงดันและกระแสคงที่ ต้องไม่เกินพิกัดที่กำหนดไว้ในสเปค

2. ความกว้างของพัลซ์โปรแกรมของขา CE (ช่วงเวลาที่ TS เป็น 1) ต้องไม่สั้นกว่า 1 มิลลิวินาที และต้องไม่นานเกิน 2 มิลลิวินาที

3. เวลาในการกำหนด (ช่วงเวลาที่ T1 เป็น 1) ต้องไมนาน เกิน 2.5 วินาที

4. เวลาพักภายหลังโปรแกรม (ช่วงเวลาที่ไม่จ่ายแรงดันหรือกระแสให้กับ PROM) ต้องไม่น้อยกว่า 2 เท่าของเวลาในการโปรแกรม เพื่อให้ได้ Duty Cycle ของการโปรแกรมน้อยกว่า 33 เปอร์เซนต์ตามที่ผู้ผลิตกำหนดมา

ที่จะยกตัวอย่างต่อไปนี้คือ EPROM ได้แก่เบอร์ 2708 ซึ่งเป็น EPROM ขนาด 1024x8 บิท ทำจาก MOSFET ชนิด Nchannel ที่มีขาเกทลอยฝังอยู่ในชั้นออกไซด์ ทำหน้าที่เก็บประจุโดยไม่ต่อกับวงจรส่วนอื่น และมีขาเกทเลือกต่อมาจากวงจรถอดรหัสเลือกแถว ทั้งนี้มันจะทำงานร่วมกับวงจรถอดรหัสเลือกคอลัม เพื่อเลือกว่า เซลหนวยจำ 8 บิทชุดไหน ถูกบังคับให้ทำงาน รูปที่ 2.8 แสดงเซลหนวย 1 เซล เมื่อยังไมถูกโปรแกรม ดังนั้น เรื่องแรงดันที่ระดับ Sense Threshold ที่

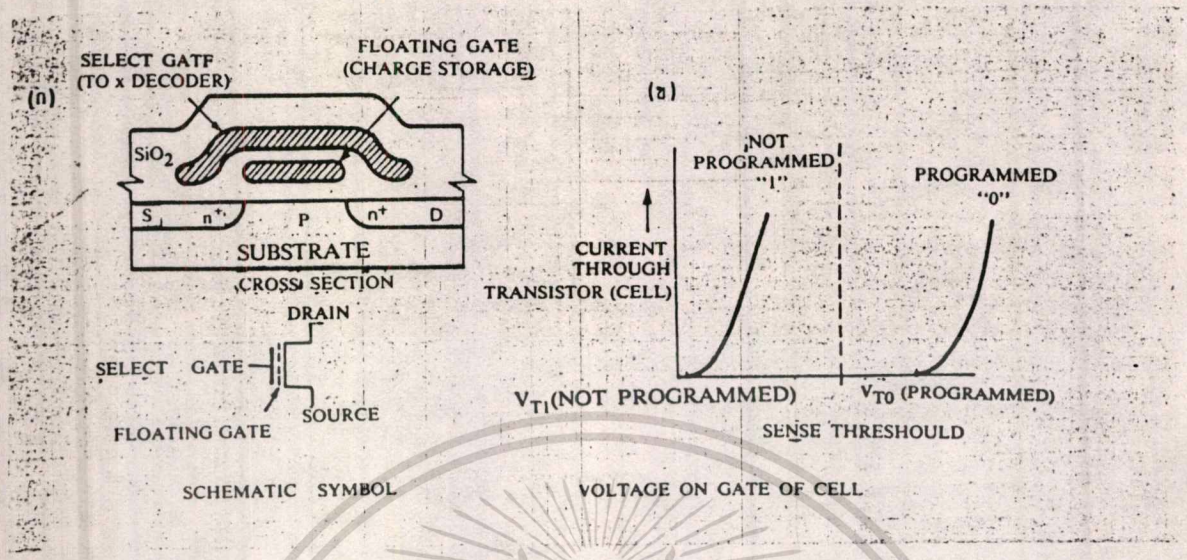
ขาเกทเลือก จะทำให้ตัวทรานซิสเตอร์สามารถนำกระแสได้ดีมาก ส่วนในกรณีที่ถูกโปรแกรม มีประจุในขาเกทลอย เซลจะมีสภาวะ เป็น 0 และกราฟลักษณะสมบัติโอนย้ายจะเป็นดังเส้นทางขวามือ ดังนั้นเมื่อป้อนแรงดันระดับ Sense Thershold ที่ขาเกทเลือก ตัวทรานซิสเตอร์ไม่สามารถนำกระแสได้ กากรดปรแกรมนั้นจึงเป็นการเลื่อน กราฟลักษณะสมบัติโอนย้ายไปทางขวามือ หรือกล่าวอีกนัยหนึ่งเป็นการเปลี่ยนค่าของตัวพารามิเตอร์ที่สำคัญอันได้แก่ Theshold Voltage VT ให้สูงขึ้น

รูปที่ 2.9 แสดงชื่อสัญญาณของขาต่าง ๆ ของ IC เบอร์นี้ รวมทั้งบล๊อคไดอะแกรมภายในด้วย A0 - A9 เป็นแอดเดรส 10 บิท ของเซลหน่วยความจำที่ต้องการเขียนหรืออ่านโดย A0 -A3 เลือกคอลัมน์ A4 - A9 เลือกแถวของหน่วยความจำ 00 -08 เป็นขั้วเข้าหรือขั้วออกของข้อมูลที่ถูกเขียนหรืออ่านออกมา CS/WE คือขาที่ใช้เลือกว่าใจะให้ IC ตัวนี้ทำงานหรือไม่ ถ้าทำงานจะทำงานในโหมดเขียนหรืออ่านออกมา ส่วนรูปที่ 2.10 แสดงการต่อขาเข้ากับแรงดันต่าง ๆ ในโมดทั้งสาม รวมทั้งแรงดันที่ปรากฏที่ขาต่าง ๆ ด้วย

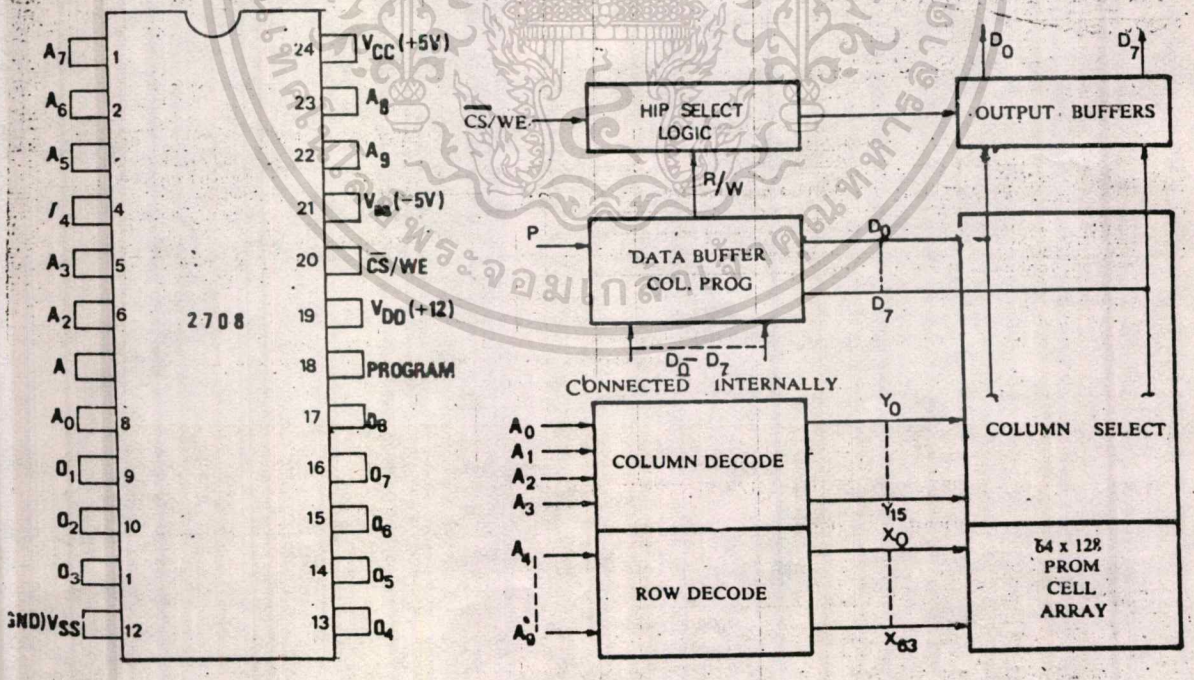
ในการใช้งาน EPROM ตัวนี้ สามารถเลือกใช้งานได้ 3 โหมด โหมดแรกคือโหมดไม่เลือก นั่นคือ 01 -08 เป็นอิมพีแดนซ์ค่าสูงโดยไม่คำนึงว่า A0 -A9 เป็นอะไร ทำให้สามารถต่อร่วมกับหน่วยความจำตัวอื่น ๆ ได้ เพื่อเพิ่มขนาดของหน่วยความจำให้มากขึ้น รายละเอียดของการ OR Tie จะได้พูดกันอีกครั้งหนึ่งในภายหลัง EPROM จะได้ทำงานในโหมดนี้ได้โดยให้ขา CS/WE มีศักดาเป็น VIH

โหมดที่สองคือ โหมดอ่าน นั่นคือ 01-08 จะให้ข้อมูลของแอดเดรส (A0 -A9) ที่เราป้อนให้กับมันออกมา 2708 ทำงานในโหมดนี้ได้โดยให้ขา CS/WE มีศักดาเป็น VIL รูปที่ 2.11แสดงรูปคลื่นของสัญญาณต่าง ๆ กรณีที่ต้องการให้ 2708 ทำงานในโหมดอ่าน คือเวลาเข้าถึงหน่วยความจำ ได้แก่ช่วงเวลาหลังจากที่ป้อนแอดเดรสเข้าไป จนถึงหน่วยความจำออกมา และก่อนที่ข้อมูลที่จะเปลี่ยนแปลง TDF คือช่วงเวลาหลังจากให้ขา CW/WE เปลี่ยนเป็น VIH ก่อนที่ข้อมูลจะเปลี่ยนแปลง

0 โหมดที่สามคือ โหมดโปรแกรม ในโหมดนี้ 01 -08 คือ ขั้วเข้าของข้อมูลที่ต้องเดรสที่ถูกกำหนดด้วย A0 - A9 อื่นที่อยู่ด้วยกัน



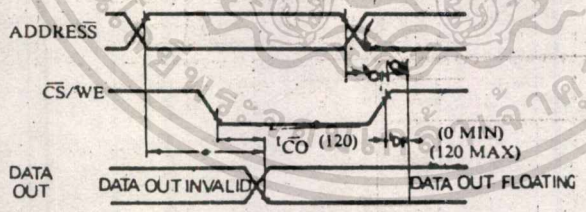
รูปที่ 2.8 (ก) เซลหน่วยจำพื้นฐานใน 2708
(ข) การเปลี่ยนค่า Threshold Voltage ของเซลหน่วยจำอันเกิดจากการโปรแกรม



รูปที่ 2.9 ชื่อสัญญาณที่ขาต่าง ๆ และบล็อคไดอะแกรมของ 2708

Function / Pin Number / Mode	Data I/O 9-11, 13-17	Address Inputs 1-7, 23,22	V_{SS} (GND) 12	Program 18	V_{DD} Supply 19	CS/WE 20	V_{BB} Supply 21	V_{CC} Supply 24
Read	D_{OUT}	A_{IN}	GND	GND	+12V	V_{IL}	−5V	+5V
Deselect	High Impedance	Don't Care	GND	GND	+12V	V_{IH}	−5V	+5V
Program	D_{IN}	A_{IN}	GND	Pulsed +26V	+12V	V_{IHW}	−5V	+5V

รูปที่ 2.10 ตารางต่อขาในโมดทั้งสามของ 2708

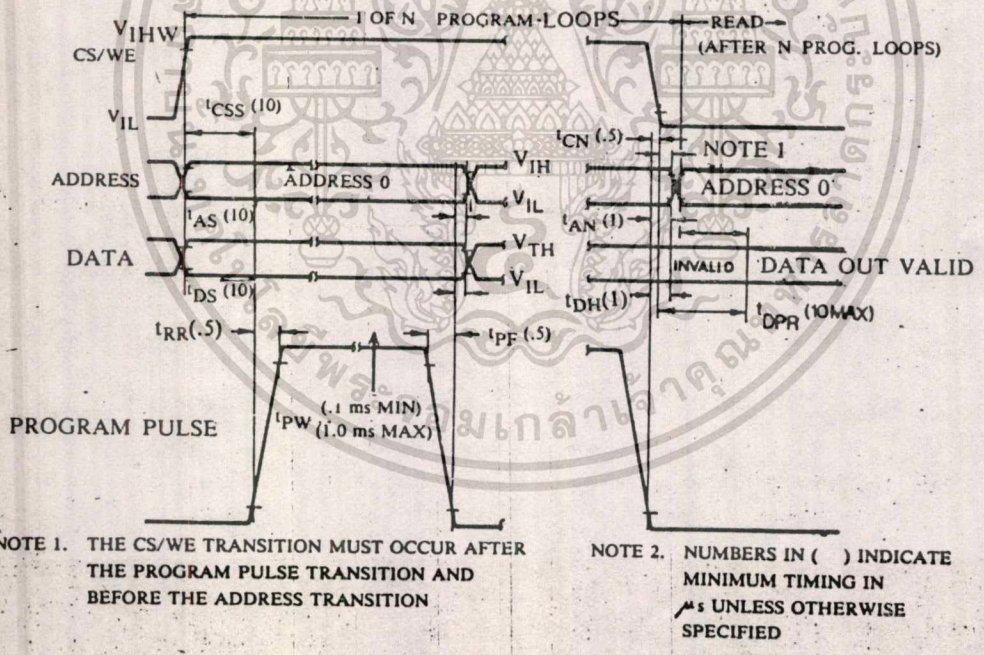


NUMBERS IN ( ) INDICATE TIMING IN NS

• t_{ACC} (280 TYP)
(450 MAX)

รูปที่ 2.11 รูปคลื่นในจังหวะอ่านของ 2708

เมื่อต้องการโปรแกรม 2708 ให้ขา CS/WE มีศักดาเป็น VIHW (12+-0.6 โวลต์) แล้วป้อนแอดเดรสให้ 2708 และป้อนข้อมูลทั้ง 8 บิท (แบบขนานกัน) ตามที่ต้องการโปรแกรมเข้าที่ขั้วเข้าตามแอดเดรสนั้น ๆ ทั้งแอดเดรสและข้อมูลมีระดับแรงดันเช่นเดียวกัน โหมดอ่านและเท่ากับระดับแรงดัน ของ TLL ขอให้ดูระดับแรงดันต่าง ๆ ในรูปที่ 2.10 พอสิ้นเวลา TAS และ TDSS ดังรูปที่ 2.12 เราถึงจะป้อนพัลซ์โปรแกรมที่มีระดับแรงดันเป็น VIHP (26+- 1 โวลต์) และพัลซ์โปรแกรมมีค่าเท่ากับ VIHP-VIL>=25 โวลต์ โปรแกรมแอดเดรสนี้เสร็จ ก็เปลี่ยนแอดเดรสแล้ว โปรแกรมต่อไปจนครบ 1024 แอดเดรสเรียกว่าครบหนึ่งรอบโปรแกรม เพื่อให้การโปรแกรมได้ผลที่น่าเชื่อถือ จะต้องโปรแกรมเช่นนี้เป็นจำนวนหลาย ๆ รอบโปรแกรม



รูปที่ 2.12 รูปคลื่นในจังหวะโปรแกรมของ 2708

Symbol	Parameter	Min.	Typ.	Max.	Units
t_{AS}	Address Setup Time	10			μs
t_{CSS}	$\overline{CS}$/WE Setup Time	10			μs
t_{DS}	Data Setup Time	10			μs
t_{AH}	Address Hold Time	1			μs
t_{CH}	$\overline{CS}$/WE Hold Time	.5			μs
t_{DH}	Data Hold Time	1			μs
t_{DF}	Chip Deselect to Output Float Delay	0		120	ns
t_{DPR}	Program To Read Delay			10	μs
t_{PW}	Program Pulse Width	1		1.0	s
t_{PR}	Program Pulse Rise Time	.5		2.0	ms
t_{PF}	Program Pulse Fall Time	.5		2.0	μs

รูปที่ 2.13 ตารางเวลาที่เกี่ยวข้องกับการโปรแกรม

ตามรูปที่ 2.12 และ 2.13 ช่วงเวลาในการโปรแกรมที่ได้ผล ดีมาก ก็คือการกำหนดค่าเวลาต่าง ๆ ดังนี้

0 TCSS = TAS = TDS = 10 ไมโครวินาที

TPW = 10 มิลลิวินาที

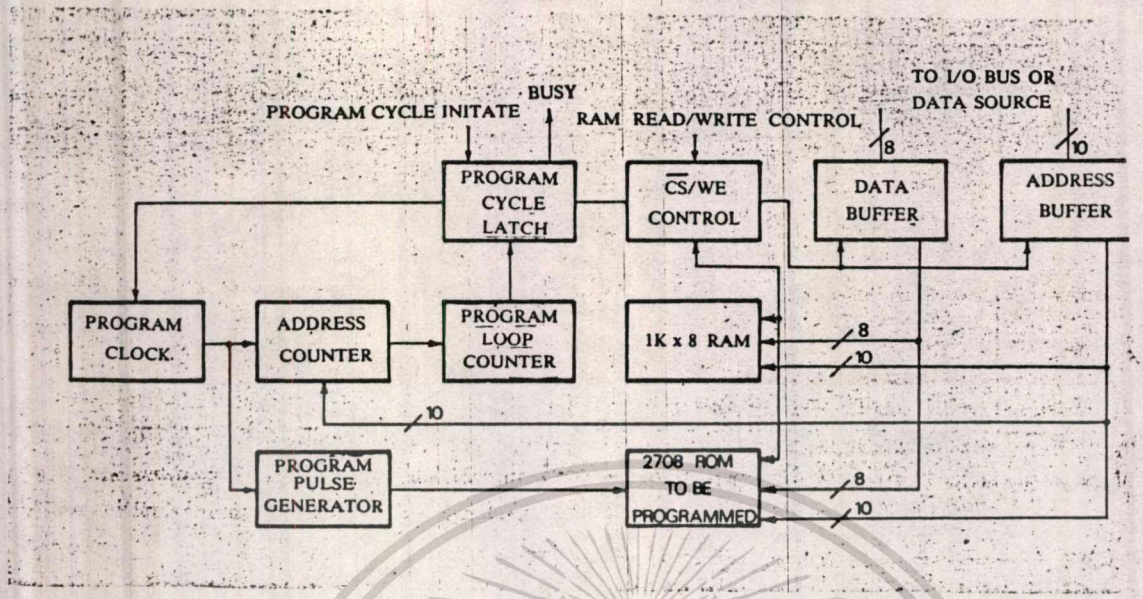
TAS = TDS = 1.0 ไมโครวินาที

TPR = TPF = 0.5 ไมโครวินาที

ดังนั้น ในการโปรแกรม 1 แอดเดรสจะใช้เวลาเป็น

TAS + TPR + TPF + TAH = 1.012 มิลลิวินาที และสำหรับ IC 1 ตัว (1024 แอดเดรสจำนวน 100 รอบ จะใช้เวลา 1.012x100x1024 = 108.6 วินาที

รูปที่ 2.14 แสดงบล็อคไดอะแกรมของชุดโปรแกรม 2708 ซึ่งมีการทำงานดังนี้ ข้อมูลทั้ง 8 บิท ผ่านบัฟเฟอร์เข้ามาเก็บใน RAM ตามแอดเดรสที่ผ่านบัฟเฟอร์เข้าไป ทั้งนี้ทั้งนั้นข้อมูลและแอดเดรสอาจมาจากระบบไมโครโปรเซสเซอร์ ที่ทำหน้าที่จ่ายข้อมูลออกมาก็ได้ หรืออาจมาจากแป้นกดข้อมูลเข้ากันได้ ครั้นข้อมูลถูกเก็บไว้ใน RAM ครบทั้ง 1024 แอดเดรสแล้ว จะมีสัญญาณสั่งเริ่มต้นโปรแกรมผ่านเข้ายัง Program Cycle Latch ซึ่งจะส่งสัญญาณ Busy กลับไปยังโปรเซสเซอร์ และยุติการส่งสัญญาณ ทั้งข้อมูลและแอดเดรส Program Cycle Latch จะส่งสัญญาณให้วงจรกำเนิดสัญญาณนาฬิกาควบคุมโปรแกรมเริ่มทำงาน และให้ RAM เริ่มทำงานด้วย ทำให้ข้อมูลจาก RAM จะเพิ่มค่าขึ้นอีก 1 แล้วการเขียนข้อมูลจาก RAM ลงใน EPROM ก็จะดำเนินต่อไปจนครบ 1024 แอดเดรสวงจร+



รูปที่ 2.14 บล็อคไดอะแกรมของเครื่องโปรแกรม 2708

ชนิดของ RAM

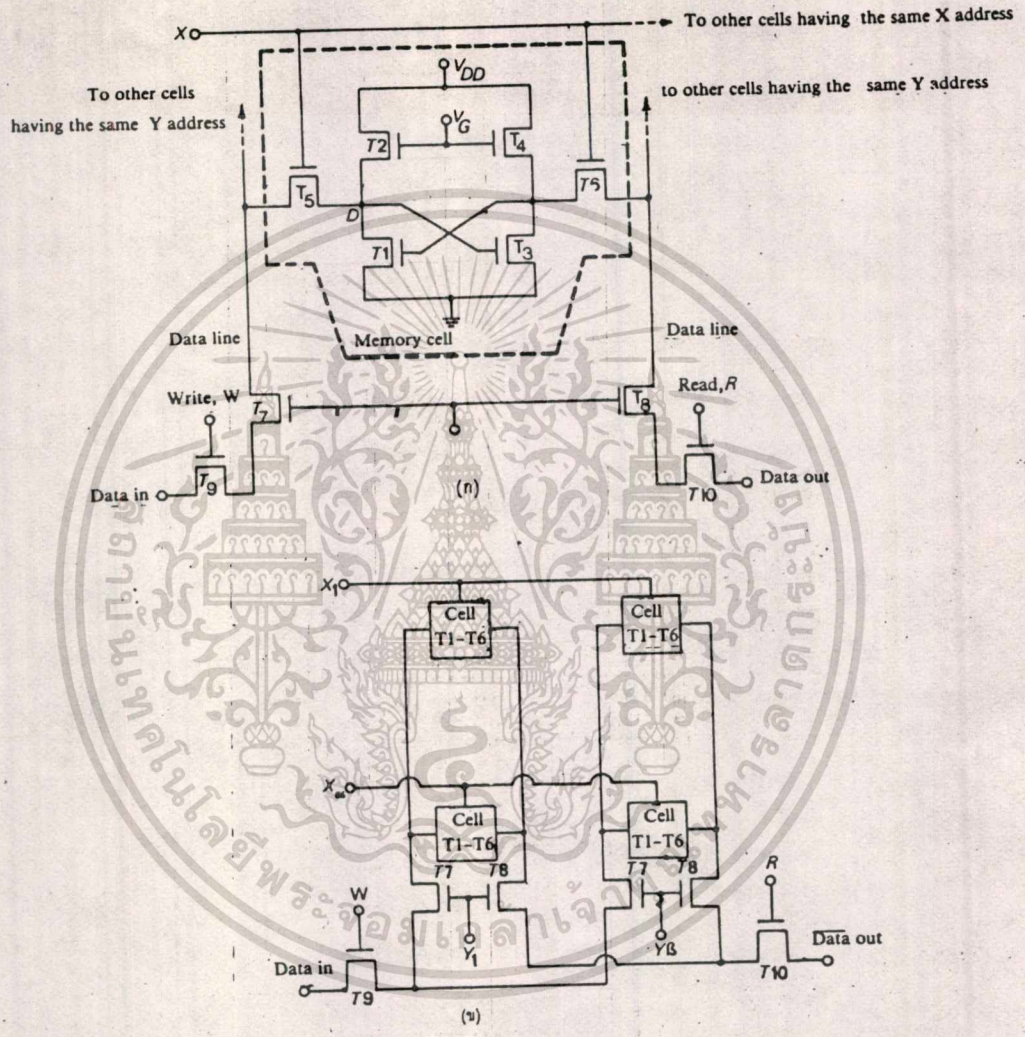
RAM ชนิดสถาติค

RAM ชนิดนี้ใช้ลักษณะวงจรฟลิปฟลอบเป็นวงจรพื้นฐานสำหรับเซลหน่วยความจำแต่ละเซลตามปกติ วงจรพื้นฐานก็มักไม่ แตกต่างกันมากนัก และโดยมากจะเป็นดังรูปที่ 2.15 ก. T1 - T6 คือทรานซิสเตอร์ที่ประกอบเป็นวงจรฟลิปในเซลพื้นฐาน ส่วน T7 -T10 เป็นเกทที่เปิดให้ข้อมูลไหลผ่านไปมาได้ เป็นวงจรร่วมของเซลหน่วยความจำหลายๆ เซล ดูรูปที่ 2.15 (ข) ดังนั้น เราจึงเรียกเซลหน่วยความจำแบบนี้ว่า เซลหน่วยความจำชนิดสถาติคแบบทรานซิสเตอร์ 6 ตัว

ลักษณะการทำงานเป็นดังนี้ T1,T2 และ T3, T4 ประกอบกันเข้าวงจรกลับเฟสที่ต่อโยงไขว้กันอยู่ ทำหน้าที่เป็นฟลิปฟลอปโดย T5,T6 เป็นเกทเปิดหรือปิดตามแต่จะเลือกด้วยระดับแรงดันในสาย X (มาจากวงจรถอดรหัสเลือกแถว) T7,T8 เป็นเกทเปิดหรือปิด ตามแต่จะเลือกด้วยระดับแรงดันในสาย Y (มาจากวงจรถอดรหัสเลือกคอลัมน์) T9,T10 เลือกว่าจะให้เขียนข้อมูลเข้าไปหรืออ่านข้อมูลออกมา

เมื่อต้องการเลือกเซลใดก็ให้สาย X,Y ของเซลนั้นเป็น 1 ทำให้สาย DATA และสาย DATA ต่อกับเซลได้ และยังต่อออกมาที่ สาย DATA IN หรือสาย DATA out ก็ได้ ด้วยการให้ W หรือ R เป็น 1 ตามลำดับ ถ้าต้องการเขียนข้อมูลเข้าไปในเซล ก็ให้ข้อมูลนั้นปรากฏที่สาย Data in และ W=1 ตามปกติเรามักให้ R=W = 0 ดังนั้น ข้อมูลจึงถูกส่งผ่าน T9, T7, T5 เข้าไปเก็บเป็นองค์ประกอบของแต่ละเซล ส่วนการทำงานยังเหมือนที่กล่าวไปแล้วเพียงแต่มีสาย X มากขึ้น เป็น 2 สาย และสาย

y มากขึ้นเป็น 3 สาย โดยมากทั้งสาย X,Y นี้ มาจากวงจรถอดรหัสเลือกแถวและคอลัมน์ ที่อยู่ในตัว IC0 และตามปกติแอดเดรสที่จะเข้ามักจะอยู่ในรูปรหัสเลขฐาน 2 n บิท การบอกขนาดของ RAM ก็เช่นเดียวกับ ROM คือบอกเป็น MxK บิท เมื่อ M คือจำนวนเวิร์ด และ K คือจำนวนบิทของแต่ละเวิร์ดหรือจำนวนบิททั้งหมดที่สามารถเข้าถึงได้ด้วยแอดเดรสเดียวกัน



รูปที่ 2.15 ก. เซลพื้นฐานของ Ram ชนิดสตาติค

ข. การต่อเซลต่าง เข้าด้วยกันภายในตัว IC

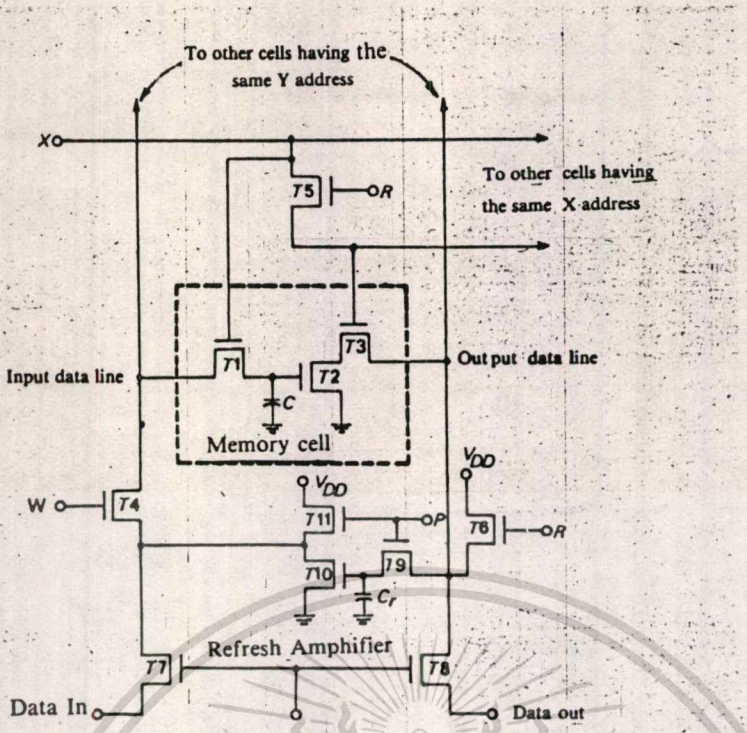
RAM ชนิดไดนามิค

RAM ชนิดนี้ใช้การเก็บประจุในตัวเก็บประจุที่ขาเกทของ MODFET แทนการเก็บข้อมูล วงจรพื้นฐานของ RAM ชนิดไดนามิคมีได้หลายรูปแบบ และใช้จำนวนทรานซิสเตอร์แตกต่างกันไป RAM ชนิดไดนามิคมีข้อดีเหนือกว่า RAM ชนิดสตาติดมากนัก นอกจากการมีเวลาเข้าถึงหน่วยความจำน้อยกว่า

ของ RAM ชนิดสตาติคเล็กน้อย แต่ต้องเสียเวลาในการรีเฟรชเท่านั้น แต่สิ่งที่น่าสนใจอย่างยิ่งก็คือการใช้จำนวนทรานซิสเตอร์ต่อ 1 เซลน้อยกว่าของ RAM ชนิดสตาติคค่อนข้างมาก ทำให้เนื้อที่ 1 ตาราง หน่วย สามารถบรรจุ RAM ชนิดไดนามิค ได้จำนวนเซลมากกว่าของ RAM ชนิดสตาติคค่อนข้างมาก ที่จริงแล้ว แต่ละเซล ของ RAM ชนิดไดนามิคอาจประกอบด้วยทรานซิสเตอร์ 1 ตัว และตัวเก็บประจุ 1 ตัว และตัวเก็บประจุ 1 ตัวเท่านั้น (คือ T1 และ C ในรูปที่ 2.16)

เมื่อจะเข้าถึงเซลใดเซลหนึ่ง เราต้องให้สาย X และ Y ของเซลนั้นเป็น 1 และแยกวงจรรีเฟรชออกจากเซลหน่วยความจำให้ P=0 ถ้าต้องการเขียนข้อมูลจากเซลหน่วยความจำ ต้องให้ R= 1, W= 0 T5, T6, T3 จะนำกระแส โดย T2 เป็นวงจรกลับเฟสของข้อมูลที่เก็บใน C ออกมาปรากฏในสาย DATA OUT

พอเวลาผ่านไปเล็กน้อย เรารีเฟรชหน่วยความจำทั้ง โดยให้ Y=0, X=1, P=1 และ R=1 T7-T8 จะหยุดกระแสข้อมูลใน C ถูกกลับเฟสโดย T2 และผ่าน T9 มาประจุให้ Cr เราเรียกช่วงขณะนี้เองว่าช่วงเวลาประจุล่วงหน้า (Precharge Period) หลังจากให้ประจุแก่ Cr เต็มที่แล้ว ก็ให้ R =0, W=1, T10 จะกลับเฟสของข้อมูลที่เก็บไว้ใน Cr ผ่าน T4, T1 กลับไปประจุให้ C ใหม่อีกครั้งหนึ่ง จะเห็นว่าข้อมูลของ C ถูกกลับเฟส 2 ครั้ง ด้วยวงจรกลับเฟส T2,T3,T8 และ T10,T11 ดังนั้นเมื่อรีเฟรชข้อมูล C โดยผ่าน T4,T1 ในการรีเฟรชหน่วยจำ เราจะรีเฟรชหน่อยความจำทั้งหมดแถว เวลาในการรีเฟรชหน่วยความจำทั้งหมดแถว จะเป็นพร้อมกัน เวลาในการรีเฟรชแต่ละแถว



รูปที่ 2.16 เซลหน่วยความจำ RAM ชนิดไดนามิคแบบทรานซิสเตอร์ 3 ตัว

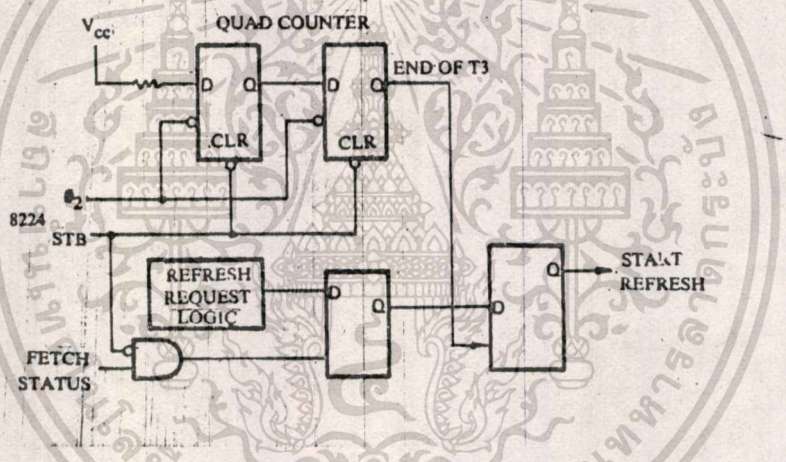
ไม่ว่าจะเป็นวิธีใดก็ตาม การรีเฟรชจะเกิดขึ้นได้ก็ต่อเมื่อหน่วยความจำนั้นต้องไม่อยู่ในระหว่างใช้งาน ทั้งนี้ การรีเฟรชอาจกระทำในลักษณะที่ซิงโครไนซ์กับสัญญาณนาฬิกาของระบบหรือไม่ซิงโครไนซ์ก็ได้

1. Asynchronous Access สัญญาณขอทำการรีเฟรชจะถูกส่งออกมาด้วยอัตราคงที่เช่น ทุก 31 ไมโครวินาที (ในกรณีมี 64 แถว) โดยไม่ขึ้นกับชนิดหรือเบอร์ของไมโครโปรเซสเซอร์ข้อเสียคือ ต้องใช้วงจรควบคุมการรีเฟรชที่ยุ่งยาก และแน่นอน ผลเสียที่ตามมาอีกอย่างหนึ่งก็คือ การมีเวลาชะลอที่ยาวนานขึ้น นอกจากนี้ สิ่งที่ต้องคำนึงถึงอีกสิ่งหนึ่งก็คือ ปัญหาความสำคัญก่อนหลัง ในการเข้าถึงหน่วยความจำ

2. Synchronous Access ซึ่งเรายังจักกันในชื่ออื่นอีกอย่างเช่น Hidden Refresh หรือ Transparent Refresh จากชื่อหลังทั้งสองนี้ ทำให้พอนึกภาพออกได้ว่า มันเป็นวิธีการรีเฟรชหน่วยความจำในขณะที่ MPU ไม่ได้เรียกใช้งาน ในการใช้งานโดยทั่วไป มักมีช่วงเวลาอยู่อย่างน้อยหนึ่งไซเคิล MPU ไม่ได้ใช้งานหน่วยความจำ และถ้าเราสามารถทราบช่วงเวลานี้ได้อย่างแน่ชัด เราก็สามารถทำการรีเฟรชหน่วยความจำได้โดยที่ MPU ไม่ได้สูญเสียประสิทธิภาพอะไร แน่นอน เมื่อ MPU ไม่มีการสูญเสียเวลาไป MPU จึงย่อมไม่รู้ว่ามีการรีเฟรชหน่วยความจำขึ้นในจังหวะนี้ วิธาการนี้มีข้อดีในเรื่องความเร็ว แต่ข้อเสียก็มีเช่นกัน ได้แก่วงจรควบคุมการรีเฟรชของไมโครโปรเซสเซอร์ชนิดใด ก็เป็นชนิดนั้น เราไม่สามารถนำไปใช้กับไมโครโปรเซสเซอร์ชนิดอื่นได้ นอกจากนี้เรายังต้องระวังเกี่ยวกับเหตุการณ์ที่ผิดธรรมดา เช่นกรณีของการใช้ไมโครโปรเซสเซอร์ 8080 A ในการใช้คำสั่ง HALT,

RESET คำสั่ง WAIT นาน ๆ ในกรณีที่ใช้หน่วยความจำที่มีความเร็วต่ำ หรือกรณีที่ทำงานแบบที่ละจังหวะ และภาวะ ในระหว่างขอ DMA ทั้งนี้การระมัดระวังจะต้องรวมไปถึงการมีวงจรลอจิก ควบคุมให้มีการโอเวอร์ไรด์เหตุการณ์นี้ เพื่อให้การรีเฟรชหน่วยความจำทุก ๆ 2 มิลลิวินาทีตามเงื่อนไข

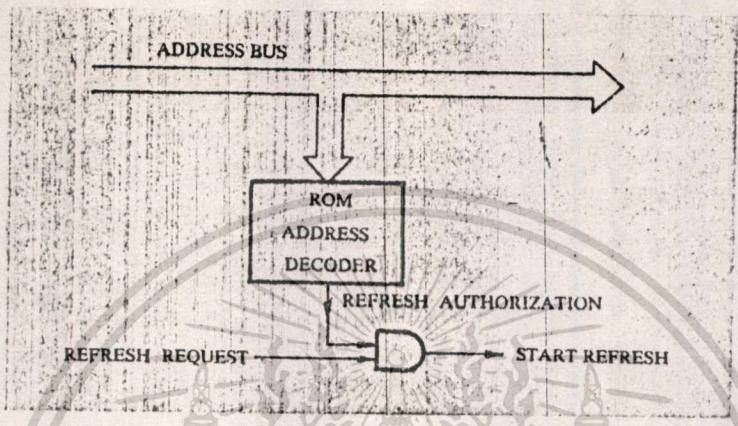
ในกรณีของไมโครโปรเซสเซอร์ 8080 A เราสามารถใช้จังหวะ T4 ของแมชีนไซเคิล M1 ในการรีเฟรชหน่วยความจำได้จึงไม่จำเป็นต้องใช้งานหน่วยความจำ รูปที่ 2.17 แสดงวงจรการรีเฟรชแบบซิงโครนัสของไมโครโปรเซสเซอร์ 8080 A ตามรูปจะมีวงจรนับสี่เป็นวงจรคอยตรวจสอบการสิ้นสุดของจังหวะ T3 เพื่อนำไปสร้างสัญญาณเริ่มต้นการรีเฟรช



รูปที่ 2.17 วงจรควบคุมการรีเฟรชแบบซิงโครนัสที่จังหวะ T4 ของ แมชีนไซเคิล M1 ของ 8080 A

เรายังสามารถรีเฟรชหน่วยความจำได้ขณะที่ใช้งาน ROM รูปที่ 2.18 แสดงจงจรควบคุมการรีเฟรชในขณะใช้งาน ROM โดยนำแอดเดรสมาถอดรหัส และตรวจสอบดูว่า เป็นแอดเดรสของ ROM หรือไม่ ถ้าเป็นก็สามารถนำไปสร้างสัญญาณเริ่มต้นการรีเฟรชได้ไมโครโปรเซสเซอร์แต่ละชนิด ยกตัวอย่างเช่น เราทราบว่าในขณะที่สัญญาณนาฬิกา 02 เปลี่ยนจากระดับ 0 เป็นระดับ 1 ตัว 8080 A ยังไม่มีความจำเป็นต้องใช้หน่วยความจำ เราก็สามารถใช้ช่วงเวลาการเปลี่ยนแปลงจากระดับ 0 เป็น

ของสัญญาณนาฬิกา 02 มากำหนดการรีเฟรชแบบไม่ชิงโครนัสได้ จึงทดทอนความยุ่งยากของวงจรได้มาก



รูปที่ 2.18 วงจรควบคุมการรีเฟรชในขณะที่ใช้งาน ROM

RAM ที่จะยกเป็นตัวอย่างในหัวข้อนี้ เป็น RAM ชนิดสถาติค เนื่องจากไม่จำเป็นต้องมีวงจรรีเฟรชและวงจำกำหนดช่วงเวลาพิเศษที่ยุ่งยากนัก ใช้ได้ง่ายและมีระดับแรงดันของสัญญาณเข้าและออกเช่นเดียวกับของ TTL อีกทั้งเป็น IC มาตรฐานที่หาซื้อในท้องตลาดได้ไม่ยากนัก หรือถ้าหาซื้อไม่ได้ ก็ยังนอใช้หลักการของ IC ตัวนี้ กับ IC ตัวอื่นที่มีขายในท้องตลาดได้ ในที่นี้จะไม่พูดถึง RAM ชนิดไดนามิคอีกต่อไป เพราะในภาคนี้ เป็นการกล่าวนำทำความเข้าใจเกี่ยวกับการใช้งานของหน่วยความจำเพื่อความสะดวกและง่ายแก่การอธิบายประกอบกันกับในระบบหน่วยความจำที่ไม่ใหญ่นัก โดยเฉพาะพวกชิงเกิลบอร์ดมักไม่นิยมใช้ RAM ชนิดไดนามิค จึงขอพูดเฉพาะ RAM ชนิดสถาติคหากท่านผู้ใดสนใจจะศึกษาเกี่ยวกับ RAM ชนิดไดนามิค ขอให้หาอ่านได้ในหนังสืออ้างอิงที่ให้ไว้ข้างท้ายบทนี้

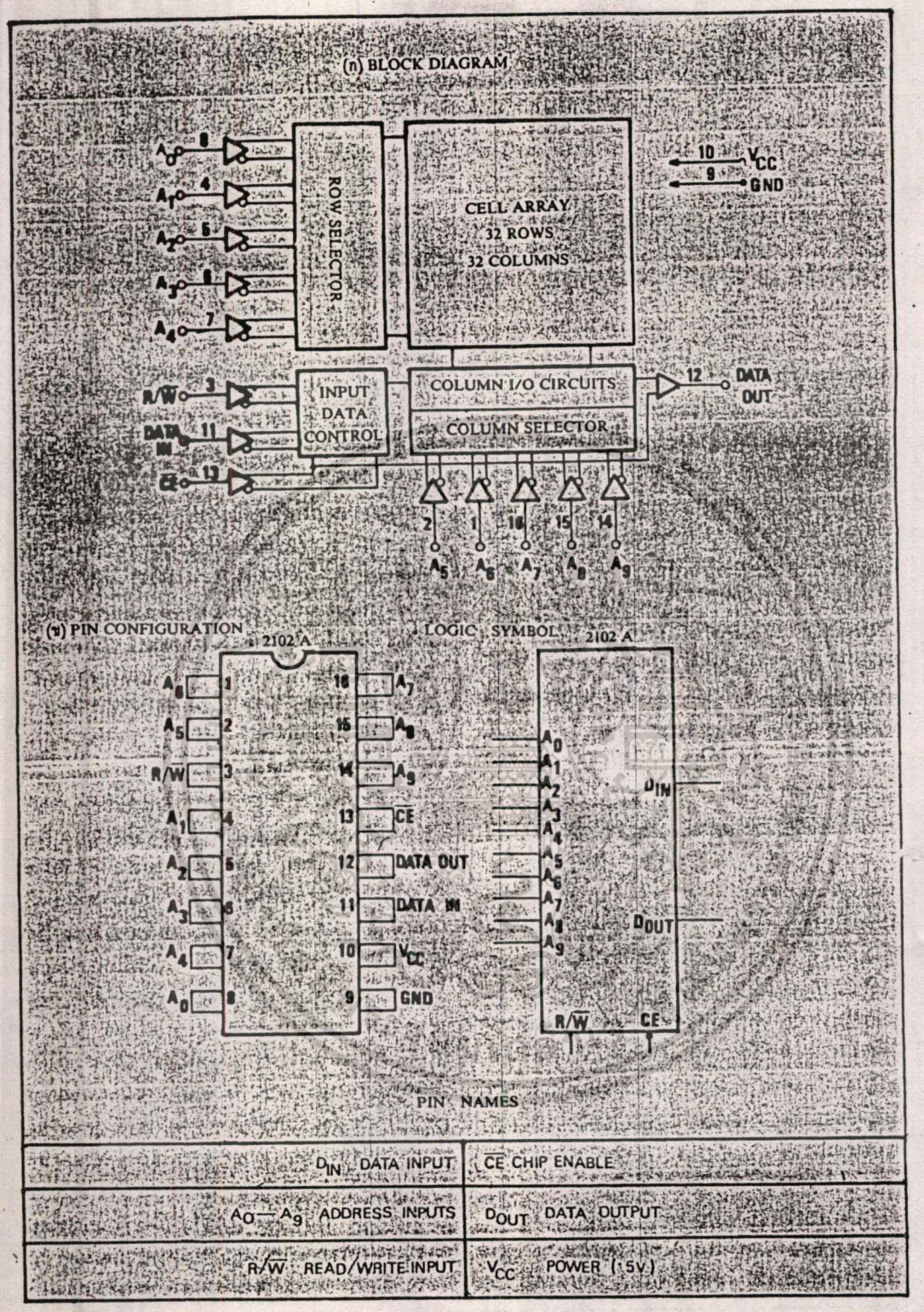
RAM ที่จะใช้ประกอบเป็นตัวอย่างในคำอธิบายต่อไปนี้ คือ 2102 ซึ่งเป็น RAM ชนิดสถาติคขนาด 1Kx1 บิท ในลักษณะของเซลพื้นฐานของหน่วยความจำหน่วยนี้ก็คล้ายคลึงกับในรูปที่ 2.15 ก. รูปที่ 2.19 แสดงบล็อคไออะแกรมขาต่าง ๆ ของ IC และสัญลักษณ์ทางตรรกของ 2102 นอกจากานี้ยังได้บอกชื่อของขาบางขาไว้ด้วย ขอให้ดูในบล็อคไดอะแกรมก่อน

รูปที่ 2.20 แสดงบ่านของข้อมูลที่เขียนเข้าไปหรืออ่านออกมา และการควบคุมให้ RAM

กลายสภาพเป็นอิมพีแดนซ์สูงข้อมูลเข้าที่ DATA in ในขณะที่ขาเลือกคอลัมน์ขาเขียนและขาชิพเอนเอเบิล เป็น 1 และเข้าบัฟเฟอร์ โดยผ่านบัสข้อมูลเข้าภายใน ในขณะที่เลือกแถวเป็น 1 เซลหน่วยความจำจะเก็บข้อมูลเข้าภายใน ในขณะที่ขาเลือกแถวเป็น 1 ในการอ่านข้อมูลจากเซลหน่วยจำ ม I/O "0" หรือสาย I/O "1" ในการอ่านข้อมูลจากเซลหน่วยความจำ มีวงจรขยายเซนซ์คอยตรวจดูว่าข้อมูลที่เก็บไว้นั้นเป็น 0 หรือ 1 แล้วส่งผ่านเกทที่ควบคุมด้วยสัญญาณชิพเอนเอเบิลสัญญาณอ่านและสัญญาณเลือกคอมมัน์ผ่านออกมาที่บัสมูลออกภายใน และผ่านบัฟเฟอร์ออกมาที่สาย DATA OUT บัฟเฟอร์สุดท้ายนี้ประกอบ MOSFET 2 ตัว ซึ่งสามารถทำให้กระแสหรือหยุดน้ำกระแส โดยสัญญาณจากบัสข้อมูลออกภายในและสัญญาณชิพเอเอเบิล ทำให้ MOSFET เปลี่ยนเป็นอิมพีแดนซ์ค่าสูง ทั้งหมดนี้ RAM แบบนี้ทำงานลักษณะ TRI STATE คือ โหมดอ่าน โหมดเขียน และโหมดอิมพีแดนซ์ค่าสูง

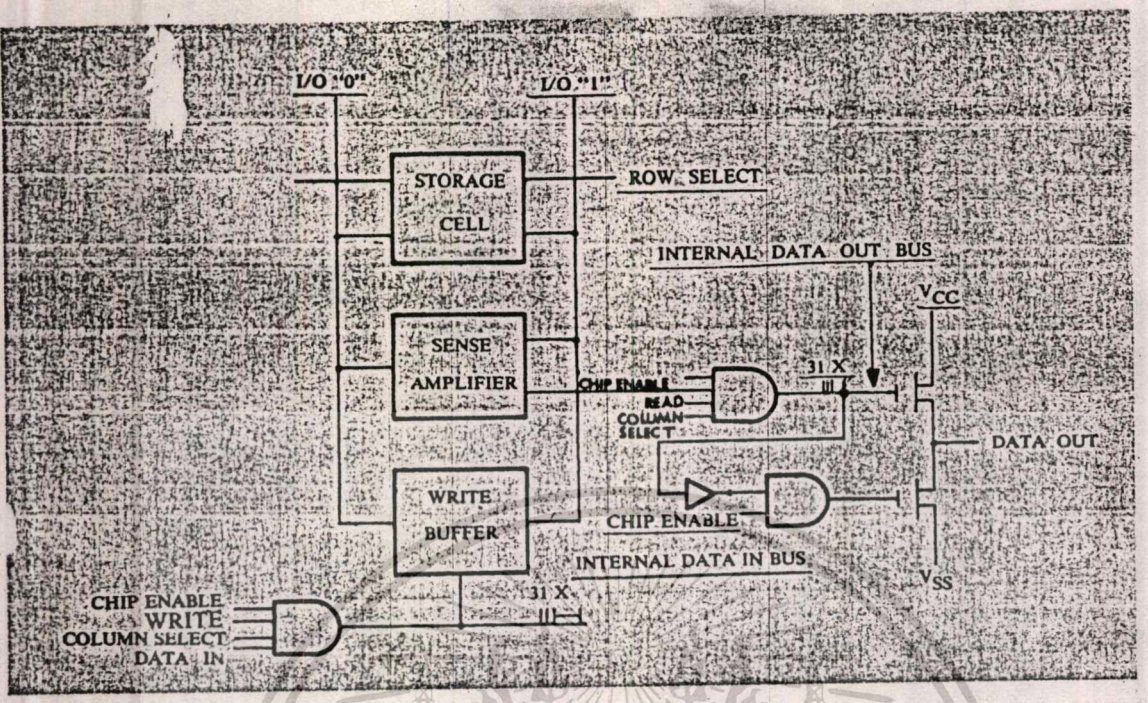
ในการอ่านหรือเขียนข้อมูลเข้าใน RAM นั้น สิ่งที่นับว่าสำคัญที่สุดก็คือ ช่วงเวลาสัมพันธ์ระหว่างสัญญาณต่าง ๆ ซึ่งจะต้องให้เหมาะสมกับ IC แต่ละเบอร์ สำหรับเบอร์ 2101 A นั้นมีรูปคลื่นของสัญญาณและช่วงเวลาสัมพันธ์ระหว่างสัญญาณต่าง ๆ ในขณะที่ต้องการอ่านและที่ต้องการเขียนดังรูป ที่ 2.21 โดยมีความหมายและขีดจำกัดของเวลาต่าง ๆ ดังปรากฏในรูปที่ 2.22

ตามรูปที่ปรากฏในรูปที่ 2.21 ก. จังหวะอ่านในขณะที่ขา R/W อยู่ในสภาวะ 1 เราต้องให้สัญญาณแอดเดรสคงที่เป็นเวลา TAC สวนสัญญาณชิพเอนเอเบิล มีได้เป็น 2 ลักษณะ ลักษณะแรก เมื่อขั้วออกของ RAM ไม่ได้ต่อแบบ OR-Tie ขา CE อาจต่อกับ GND ได้โดยตรง ทำให้ RAM ตัวนี้ทำงานตลอดเวลา ดังนั้น ข้อมูลสามารถปรากฏที่ขั้วออกได้ภายในเวลา TA ลักษณะที่ สอง เมื่อขั้วออกของ RAM ต่อแบบ OR-Tie อยู่ ขา CE สามารถเปลี่ยนสภาวะเป็น 0 เมื่อใดก็ได้ เพื่อให้ข้อมูลปรากฏที่ขั้วออก ถ้าหากมีเปลี่ยนสภาวะของสัญญาณชิพเอนเอเบิลก่อนเลือกแอดเดสร หรือหลังจากเลือกแอดเดรสไปเป็นเวลาไม่เกิน TA -TCO ขั้วออกจะมีข้อมูบปรากฏภายในเวลา TCO นับจากเลาที่มีการเปลี่ยนสภาวะของสัญญาณชิพเอนเอเบิล หากมีการเปลี่ยนแอดเดรสใหม่ ข้อมูลของแอดเดรสเดิมจะยังคงค้างอยู่เป็นเวลา TOH1 หรือในกรณีที่เปลี่ยนสภาวะของสัญญาณชิพเอนเอเบิลเป็น 1 ข้อมูลของแอดเดารเดิมจะยังคงค้างอยู่เป็นเวลา TOH2 ดังนั้น ในการออกแบบระบบหน่วยความจำ เราต้องการอ่านข้อมูลออกมาจากหน่วยความจำ เราต้องคำนึงถึงช่วงเวลาที่ต้องรอ เพื่อให้ได้ข้อมูลออกมาหลังจากเลือกแอดเดรสและสัญญาณเอนเอเบิลไปแล้ว และช่วงเวลาที่ข้อมูบแอดเดรสยังคงค้างอยู่ด้วย

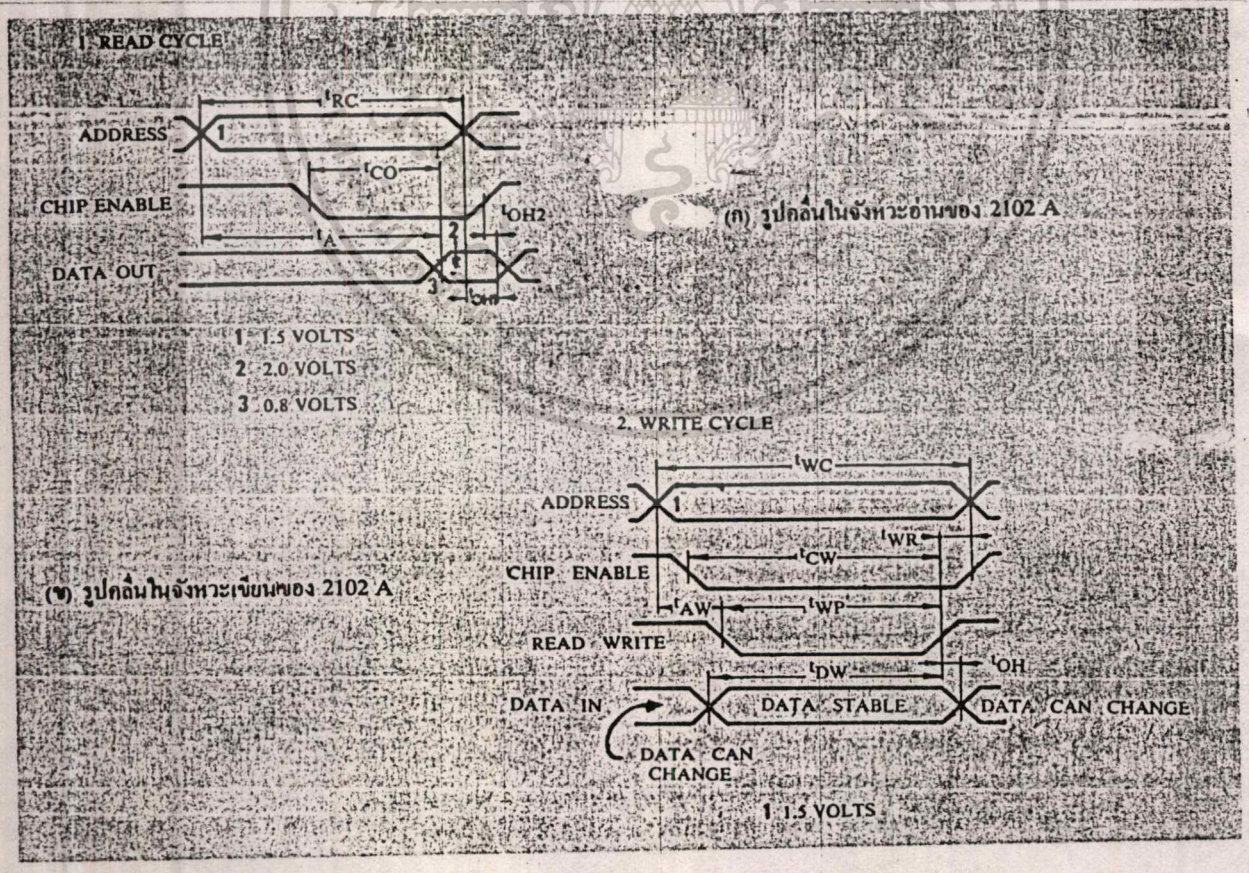


รูปที่ 2.19 ก. บล็อคไดอะแกรมภายในของ 2102 A

ข. ชื่อสัญญาณที่ขาต่าง ๆ และสัญลักษณ์ทางตรรกของ 2120 A



รูปที่ 2.20 ทางผ่านของข้อมูลใน RAM



รูปที่ 2.21

1.2102A Read Timing

READ CYCLE

Symbol	Parameter	Min.	Typ[1]	Max	Unit
t_{RC}	Read Cycle	350			ns
t_A	Access Time			350	ns
t_{CO}	Chip Enable to Output Time			180	ns
t_{OH1}	Previous Read Data Valid with Respect to Address	40			ns
t_{OH2}	Previous Read Data Valid with Respect to Chip Enable	0			ns

2. 2102A Write Timing

WRITE CYCLE

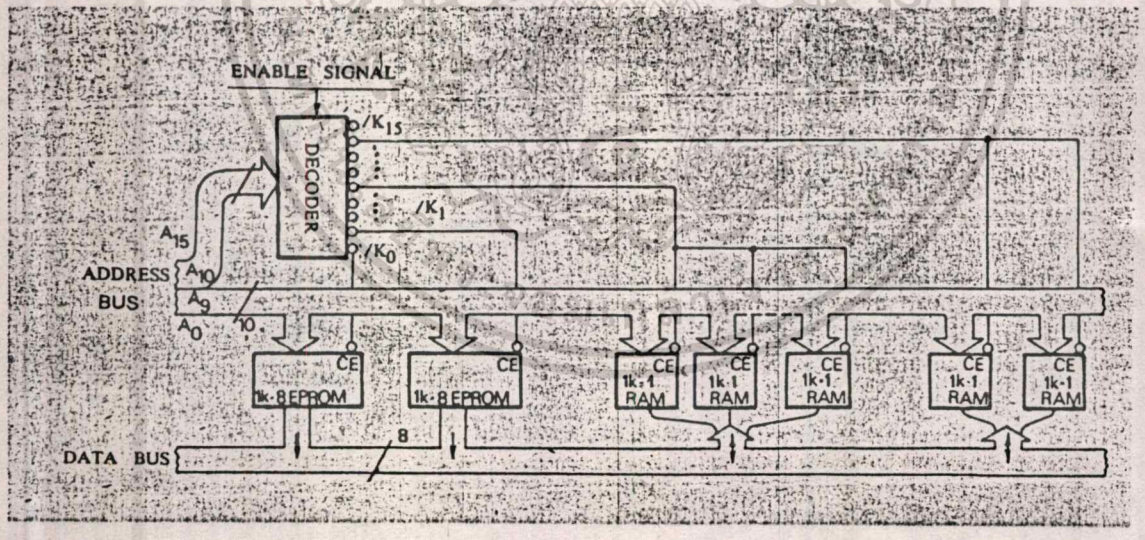
Symbol	Parameter	Min.	Typ[1]	Max	Unit
t_{WC}	Write Cycle	350			ns
t_{AW}	Address to Write Setup Time	20			ns
t_{WP}	Write Pulse Width	250			ns
t_{WR}	Write Recovery Time	0			ns
t_{DW}	Data Setup Time	250			ns
t_{DH}	Data Hold Time	0			ns
t_{CW}	Chip Enable to Write Setup Time	250			ns

NOTE: 1 Typical values are for T_A = 25C and nominal supply voltage

รูปที่ 2.22 1. ข้อมูลเกี่ยวกับช่วงเวลาในจังหวะอ่านของ 2102 A

2. ข้อมูลเกี่ยวกับช่างในจังหวะเขียน ของ 2102 A

รูปที่ 2.23 แสดงการต่อระบบหน่วยความจำที่ใช้ EPROM ขนาด 1Kx8 บิท จำนวน 2 ตัวและ RAM ขนาด 1Kx1 บิท จำนวนหลาย ๆ ตัว ในบัสแอดเดรส มีทั้งหมด 16 สาย จึงสามารถต่อกับหน่วยความจำได้ ทั้งนี้รวมทั้งของ EPROM และ RAM (ในที่นี้แต่ละเวิร์ดมี 8 บิท) ในบัสข้อมูล มีทั้งหมด 8 สาย ตามปกติ IC 1 ตัว ไม่สามารถมีจำนวนเวิร์ดและจำนวนบิทได้มากตามต้องการ เช่นนี้จึงต้องใช้ IC หลายตัว และเนื่องจากข้อมูลจาก IC แต่ละตัวต้องมาปรากฏในบัสข้อมูล ซึ่งเป็นสายสัญญาณร่วม จึงนิยมใช้วิธีต่อขั้วออกเข้าด้วยกันในลักษณะ OR-TIE นั่นคือ D0 ของ IC ทุกตัวต่อถึงกันหมด D1 ก็เช่นกันจึงถึง D7 ทีนี้ในการใช้งาน เมื่อต้องการเข้าถึงเซลหน่วยความจำใน IC ตัวไหน เราเลือกให้ IC ตัวนั้นทำงาน โดยให้ขา CE ของ IC ตัวนั้นเป็นสภาวะ 0 ในขณะที่ตัวอื่นไม่ทำงาน (ขา CE เป็นสภาวะ 1) ข้อนี้ต้องการการระวังเป็นพิเศษ มิฉะนั้นข้อมูลที่ได้มาอาจผิดพลาด

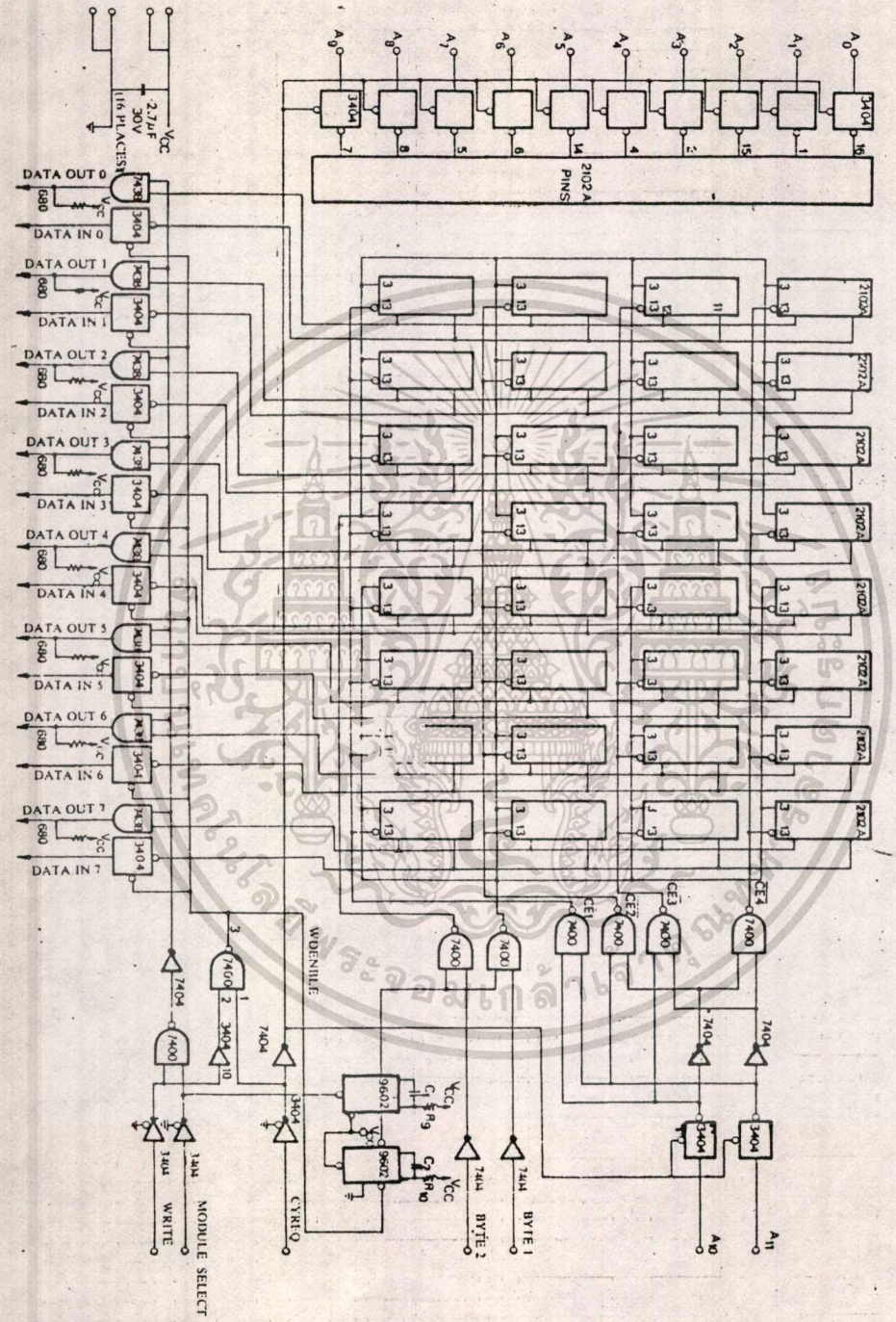


รูปที่ 2.23 ระบบหน่วยความจำขนาด 8 บิทที่ประกอบจาก EPROM (2708) และ RAM (2102 A)

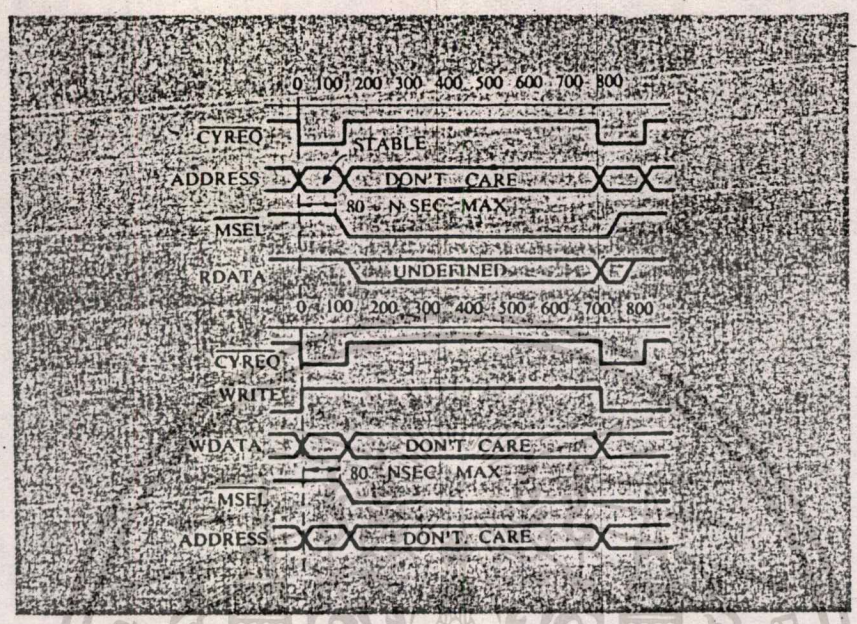
รูปที่ 2.24 เป็นระบบหน่วยความจำขนาด 4K X8 โดยมีรูปคลื่นของสัญญาณต่าง ๆ ดังในรูปที่ 2.25 หน่วยความจำพื้นฐานที่ใช้ก็คือ 2102A โดยมีสายแอดเดรส A0-A9 ผ่านแลทช์ความเร็วสูง (3404) มายังแอดเดรสทั้งหมดของ IC ทุกตัว เนื่องจากเป็นระบบหน่วยความจำขนาด 4K เวิร์ด จึงต้องมีวงจรถอดรหัส ซึ่งประกอบด้วย 3404,7400 ทำหน้าที่แปลง A10-A11 เป็น 4 สาย วงจรถอดรหัสจาก 2 สายเป็น 4 สาย มีขั้วเข้าและขั้วออกของข้อมูลแยกจากกัน โดยผ่าน 3404 และ 7438 ตาม ลำดับ วงจรดังในรูปที่ 2.24 นี้เรียกว่าเป็น 1 โมดุล อาจประกอบกันอยู่ในแผ่นวงจร 1 แผ่น วงจร 1 แผ่น ในบางระบบอาจมีหน่วยความจำแบบนี้หลายแผ่น จึงต้องมีสายเลือกโมดุล เลือกว่าเป็นแผ่นวงจรแผ่นใด มีสาย Byte 1 และ Byte 2 เลือกว่าจะทำอะไรกับข้อมูลใน 4 บิทต้นหรือ 4 บิท ท้ายได้ โดยมีการควบคุมอีกชั้นหนึ่งจากสายสัญญาณเขียนซึ่งผ่าน 3404, 7400,9602 (Retriggerable Monostable) และ 2 เกทสุดท้าย คือ 7400 ทั้งสองตัวก่อนที่จะเข้าไปยังขา R /W ของ2102 A แต่ทั้งหมดนี้จะทำงานไม่ได้ หากไม่มีสัญญาณ Cycle Request (สาย CYREQ)

ขอให้ดูรูปที่ 2.25 ที่เวลา T=0 หน่วยเวลาในรูปคือนาโนวินาที ป้อนพัลซ์ CYREQ ขนาดกว้าง 100 นโนวินาทีเข้ามาขณะที่แอดเดรสอยู่คงที่ตามต้องการ และถูกเก็บในแทช์ (3404) แอดเดรสคงที่เป็นเวลา 100 นาโนวินาที สาย MSEL จะต้องเปลี่ยนคงที่แล้ว และในกรณีที่ต้องการเขียนข้อมูลเข้าไปในหน่วยความจำสายที่ป้อนสัญญาณเขียนจะต้องเป็น 1 เป็นเวลา 650 นาโนวินาทีหน่วยความจำ ข้อมูลจะปรากฏที่ขั้วออก ในเวลาไม่นานหลังจากที่สาย MSEL เปลี่ยนเป็นสภาวะ 1 ในกรณีที่ใช้แผ่นวงจรหน่วยความจำเพียงแผ่นเดียว 1 โมดุล สายเลือกโมดุลอาจต่อกับ GND ได้เลย

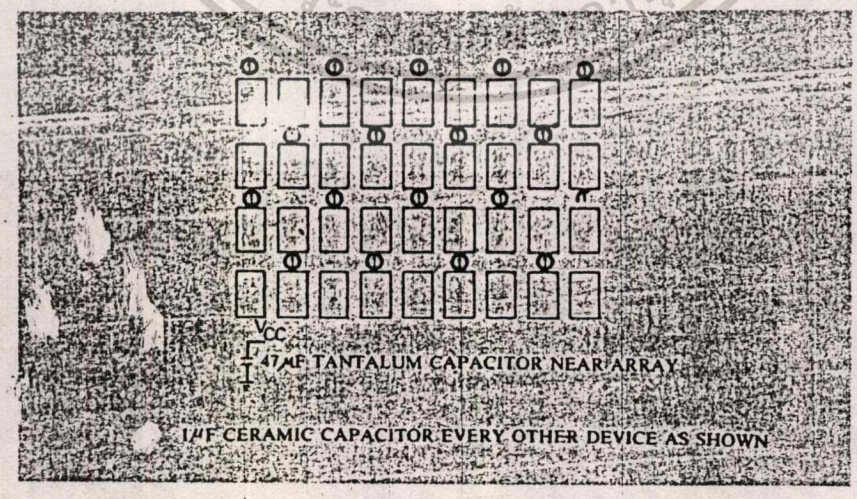
เพื่อป้องกันการรบกวนอันเนื่องจากการเชื่อมโยงผ่านทาง สาย VCC และ GND จึงต้องมีการต่อตัวเก็บระจุขนาดเล็ก ลงในแผ่นวงจรหน่วยความจำด้วย เพื่อกรองสัญญาณรบกวนในสายจ่ายไฟตรง ดังเช่นการต่อตัวเก็บประจำขนาด .1 ลงในตำแหน่งที่หมายด้วยเลข (1) ในรูป เราเรียกวิธีการนี้ว่าการคัปเปิลในระบบหน่วยความจำ



รูปที่ 2.24 ระบบหน่วยความจำขนาด 4K X8 ของ 2102 A



รูปที่ 2.25 รูปคลื่นและช่วงเวลาต่าง ๆ ที่เกี่ยวข้องในระบบหน่วยความจำ ขนาด 4K X8



รูปที่ 2.26

บทที่ 3
ไมโครโปรเซสเซอร์ Z-80

3.1 บทนำ

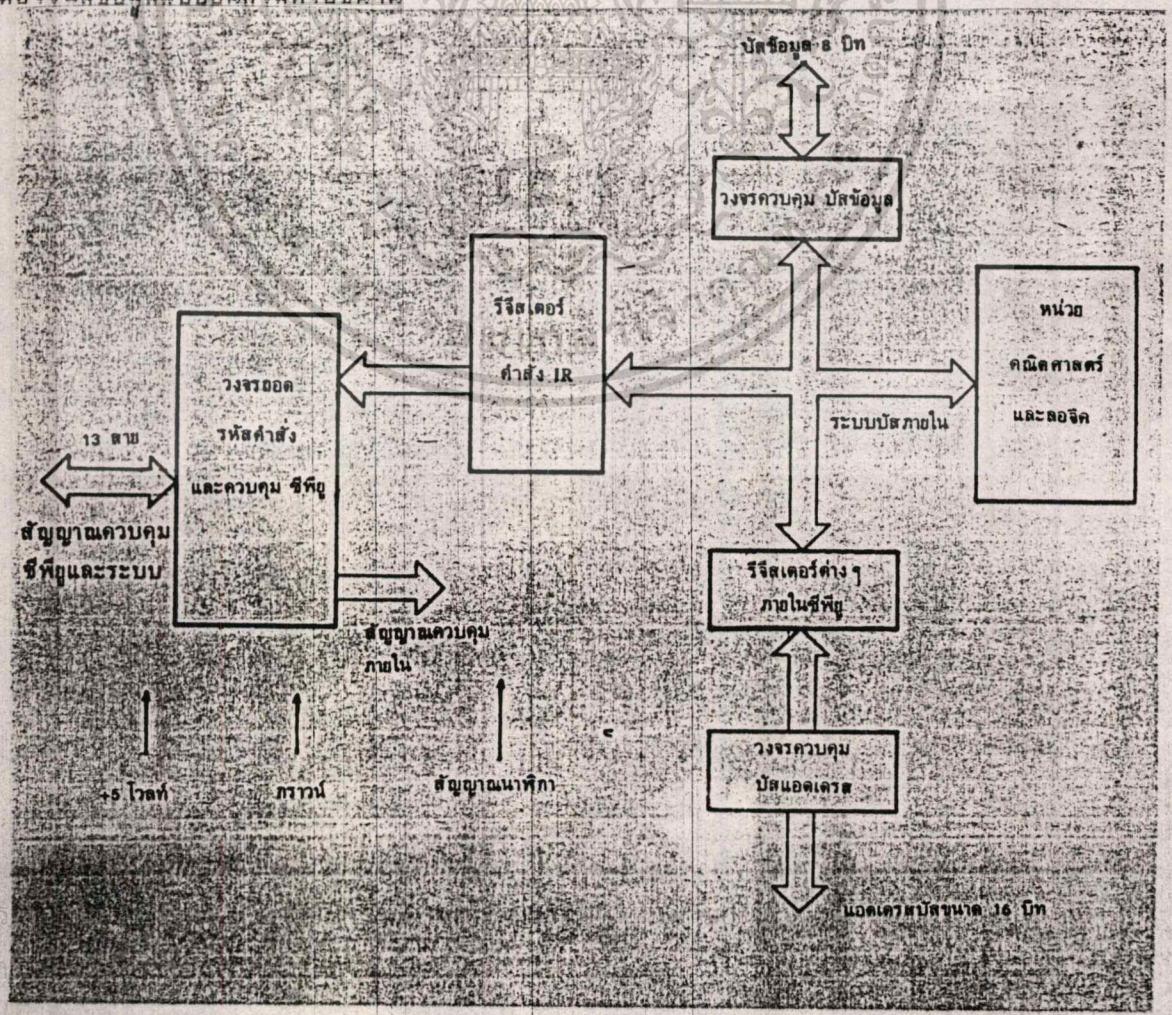
หลังจากที่ไมโครโปรเซสเซอร์เบอร์ 8080 ได้ประสบผลสำเร็จในด้านการขายและการพัฒนาระบบไมโครคอมพิวเตอร์ทางด้านซอฟท์แวร์ได้เจริญก้าวหน้าจนทำให้เครื่องไมโครคอมพิวเตอร์ที่ใช้ ซีพียู 8080 สามารถทำงานได้กว้างขวางจนถึงระดับภาษาชั้นสูง เช่น BASIC,FORTRAN ฯลฯ นอกจากนี้การพัฒนาระบบการจัดการทำงาน (Operating system) ที่ใช้ร่วมกับอุปกรณ์เพอริเฟอรัลต่าง ๆ ก็ได้ก้าวมาถึงจุดที่ทำให้เครื่องไมโครคอมพิวเตอร์ 8080 เป็นระบบที่สมบูรณ์ได้เป็นอย่างดี

เพื่อให้ประสบผลสำเร็จทางด้านการตลาดในการขายไมโครโปรเซสเซอร์ บริษัทผู้ผลิตจึงต้องคำนึงถึงระบบซอฟแวร์ที่มีการพัฒนาไปมากแล้ว วิธีการหนึ่งก็ด้วยการสร้างไมโครโปรเซสเซอร์เบอร์ใหม่ขึ้นมาที่มีโครงสร้างสถาปัตย์กรรมดีกว่าตัวเดิมที่มีอยู่แล้ว แต่คงใช้ชุดคำสั่งเดิมได้เพื่อจะได้ใช้กับระบบซอฟแวร์ที่มีอยู่แล้ว Z-80 เป็นไมโครโปรเซสเซอร์ที่มีจุดเริ่มต้นมาจากการพัฒนาและสร้างขึ้นของทีมวิศวกรบริษัทไซล็อก (Zylox) Z-80 เป็นไมโครโปรเซสเซอร์ขนาด 8 บิท ที่มีลักษณะน่าสนใจบางประการดังนี้

1. Z-80 มีลักษณะทางซอฟท์แวร์ที่สามารถนำไปใช้แทนไมโครโปรเซสเซอร์เบอร์ 8080 ได้

2. Z-80 มีลักษณะนิเศษทางฮาร์ดแวร์หลายประการ เช่น มีโครงสร้างที่มีความสามารถสร้างรวมอยู่ในชิปเดียว ใช้อัตราของสัญญาณนาฬิกาสูงถึง 4 MHz ใช้แหล่งจ่ายไฟเลี้ยงเพียงชุดเดียวคือ 5 โวลท์ และต้องการสัญญาณนาฬิกาเพียงเฟสเดียว

3. มีอุปกรณ์ที่เป็นชิปประกอบอีกที่หาได้ง่าย เช่น ชิปที่ใช้ติดต่อกับเพอริเฟอรัลต่าง ๆ ที่ใช้ในการอินเตอร์เฟสข้อมูลแบบอนุกรมหรือขนาน

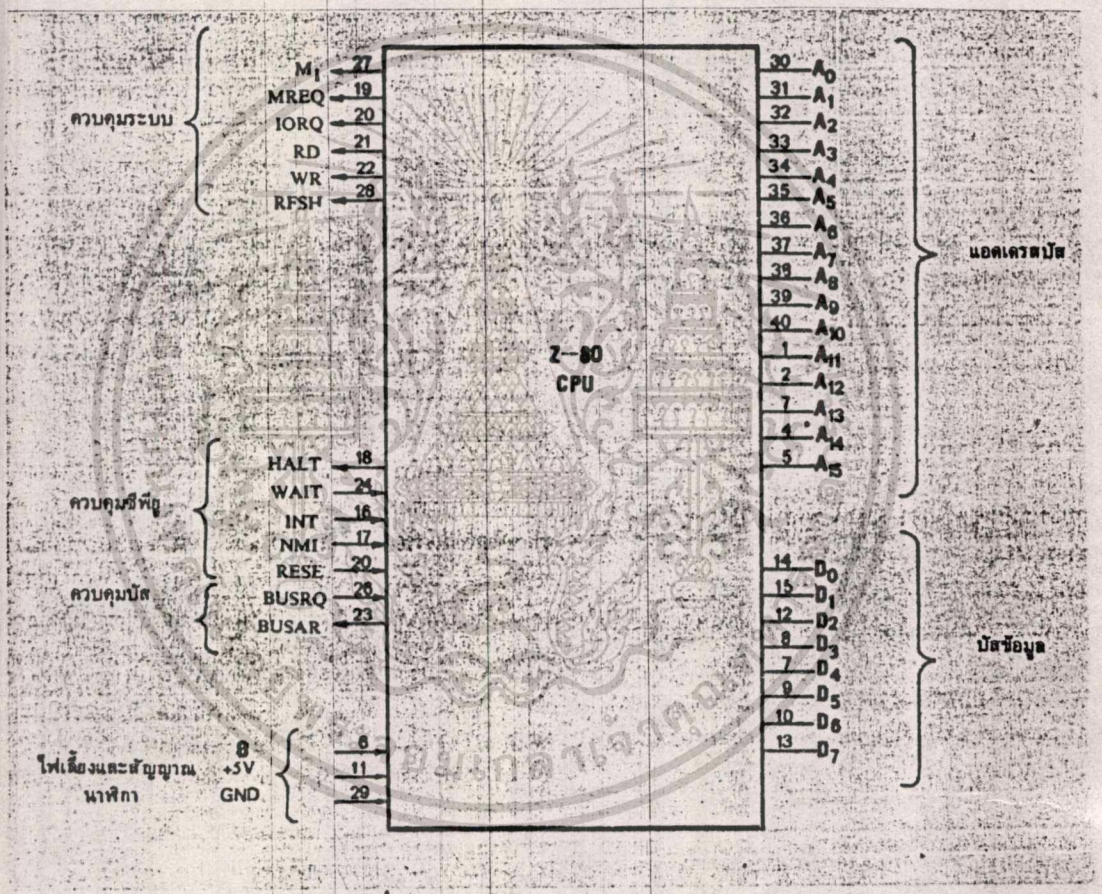


รูปที่ 3.1 โครงสร้างภายในของซีพียู Z-80

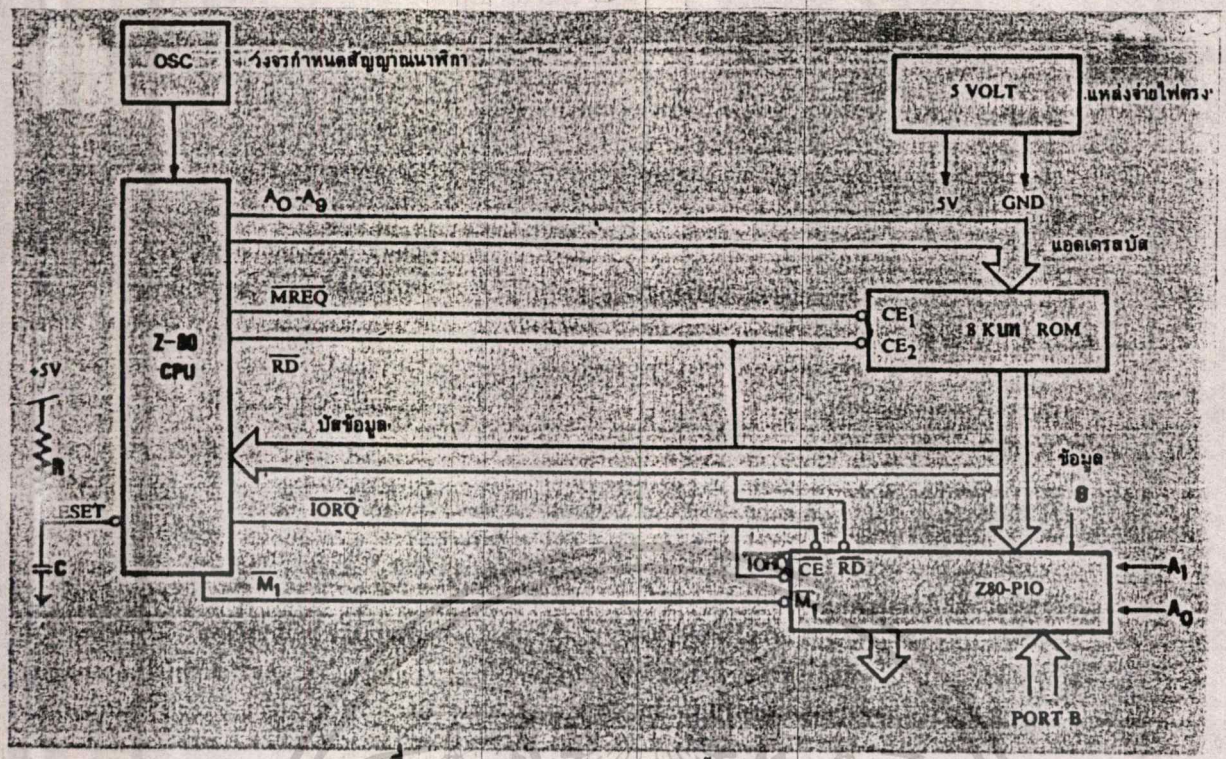
3.2 โครงสร้างทั่วไปของ ซีพียู

โครงสร้างทางซีพียู (Z-80) ก็ลักษณะคล้ายคลึงกับของ 8080 มาก จะมีข้อแตกต่างกันก็ในรายละเอี
ที่ Z-80 มีมากกว่าและเกินมาจากของ 8080 ลักษณะโครงสร้างของ Z-80 แสดงให้เห็นดังรูปที่ 3.1

โครงสร้างภายในของซีพียูจะประกอบด้วยบัสข้อมูล 8 บิทซึ่งเป็นบัสชนิดสองทิศทางคือ ข้อมูลสามารถ
เข้าหรือออกจากซีพียูได้ บัสแอดเดรสเป็นบัสขนาด 16 บิท ที่จะทำให้มีความสามารถในการอ้างถึงแอดเด
ได้โดยตรงได้ถึง 2^{16} = 64K บัสแอดเดรสนี้เป็นสายสำคัญในการอ้างถึงแอดเดรสของหน่วยเนอริเฟอรัลที่
อินพุทหรือเอาท์พุทด้วย สายบัสควบคุมนั้นจะประกอบไปด้วยสัญญาณควบคุมต่าง ๆ ซึ่งมีอยู่ด้วยกันทั้งหมด 13
ลักษณะการจัดขาไอซีแสดงให้เห็นดังรูปที่ 3.2



รูปที่ 3.2 การแสดงการจัดขาของ Z-80



รูปที่ 3.3 ระบบไมโครคอมพิวเตอร์พื้นฐานของ Z-80

รายละเอียดของขาต่าง ๆ และหน้าที่สำคัญมีดังนี้

A_0-A_{15} เป็นสายแอดเดรสบัส 16 สายที่ส่วนภายในของเอาท์พุทเป็นลอจิค 3 สถานะ (tristate output ที่จะถูกอินาเบิ้ลเลือกว่าเวลาใดเป็นสัญญาณใด ทั้งนี้เพราะสายของแอดเดรสยังทำหน้าที่เป็นตัวอ้างแอดเดรสสำหรับอุปกรณ์ I/O อีกด้วย

D_0-D_7 เป็นสายของบัสข้อมูลจำนวน 8 สาย ลักษณะของขานี้เป็นลอจิค 3 สถานะสองทิศทาง (tristate input/output) เพื่อเลือกทิศทางการไหลของข้อมูลระหว่างซีพียูกับหน่วยความจำหรืออุปกรณ์อินพุทเอาท์พุท

M_1 ลักษณะจะเป็นเอาท์พุทโดยส่งสัญญาณออกมาเพื่อบอกให้ทราบว่ากำลังอยู่ในสภาวะเฟทช์ โดยแอคทีฟที่ลอจิค "0"

MERQ เป็นเอาท์พุทลอจิค 3 สถานะ สัญญาณเอาท์พุทขานี้จะเป็นตัวบอกว่า ขณะนี้สัญญาณที่แอดเดรสบัสมีค่าแอดเดรสเพื่อเขียนหรืออ่านในหน่วยความจำการแอกตีฟจะแอกตีฟที่ลอจิค "0"

IORQ เป็นเอาท์พุทที่ให้สัญญาณเพื่อบอกว่าขณะนี้สัญญาณในแอดเดรสบัสจาก A_0-A_7 มีค่าแอดเดรสของ I/O อยู่ ประโยชน์ของสัญญาณนี้เพื่อดีเทคหัสแอดเดรสในการเขียนหรืออ่านข้อมูลจากเนอริเฟอรัล

RD เป็นสัญญาณเอาท์พุทที่จะบอกให้ทราบว่า ขณะนี้ซีพียูต้องการอ่านข้อมูลจากหน่วยความจำหรือ I/O

เป็นข ...ที่ให้สัญญาณเอาท์พุทเมื่อจะบอกว่าขณะนี้ ซีพียูต้องการจะเข้าถึงข้อมูลในหน่วยความจำหรือ I/O

SM เป็นสายที่ส่งสัญญาณที่จะบอก... ขณะนี้สายของบัสแอดเดรสจะมีข้อมูลแอดเดรสสำหรับการรีเฟรช หน่วยค... ความจำแบบไดนามิก

ALT เป็นสายที่จะแอกตีฟเมื่อซีพียูกระทำคำสั่ง HALT โดยจะแอคตีฟให้ลอจิค "0"

AIT เป็นสัญญาณที่จะบอกให้ทราบว่าขณะนี้ หน่วยความจำหรือ I/O ยังไม่พร้อมที่จะรับหรือส่งผ่านข้อมูล คือเมื่อส่งสัญญาณนี้เข้าไป ซีพียูจะหยุดรอจนกว่าจะเลิกสัญญาณ wait

NT เป็นสัญญาณจาก I/O ที่จะอินเทอร์รัพท์ซีพียูการอินเตอร์รัพท์จะมีหลายโหมดเป็นแบบมาสเคเบิ้ล (maskable interrupt)

ESET เป็นสัญญาณที่จะส่งเข้าไปรีเซทซีพียูหรือทำให้โปรแกรมเคาว์เตอร์มีค่าเป็น 0

MI เป็นสัญญาณอินเทอร์รัพท์ที่เป็นนอนมาร์คเคเบิ้ล อินเตอร์รัพท์ (non maskable interrupt)

USRQ เป็นการส่งสัญญาณบอกซีพียูว่า ขณะนี้จะต้องการใช้บัสจะทำให้ซีพียูกวบคุมบัส โดยใช้หลักการลอจิคสามสถานะ ในการทำให้บัสแอดเดรสและบัสข้อมูลแยกออกจากระบบในซีพียูเมื่อให้หน่วยความจำและ I/O ใช้บัสในการเคลื่อนย้ายข้อมูลระหว่างกัน

USAR เป็นสัญญาณจากซีพียูที่จะส่งออกไปบอกว่าขณะนี้ซีพียูไม่ได้ใช้บัสแล้ว

ายเหตุ การใช้-อยู่เหนือสัญญลักษณ์แสดงว่า เป็นการแอกตีฟที่ลอจิค "0"

3.3 ระบบทางฮาร์ดแวร์ของไมโครคอมพิวเตอร์ Z-80

วงจรฮาร์ดแวร์พื้นฐานของไมโครคอมพิวเตอร์ Z-80 ประกอบด้วย

แหล่งจ่ายไฟตรงขนาด 5 โวลท์

วงจรกำเนิดสัญญาณนาฬิกา

อุปกรณ์หน่วยความจำ RAM หรือ ROM

.วงจร I/O

.ซีพียู Z-80

ตัววงจรประกอบขึ้นด้วยสิ่งต่างๆ เหล่านี้เขียนเป็นไดอะแกรมได้ดังรูปที่3

ากระบบเราจะเห็นว่า ซีพียูยังต้องการแหล่งจ่ายไฟขนาด +5 volts กับกราวนด์

ละวงจร กำเนิดสัญญาณนาฬิกา ส่วนการต่อหน่วยความจำนั้น (รูปใช้ ROM)

จะต่อแอดสบัสเข้าโดยตรงกับแอดเดรสของหน่วยความจำและบัสข้อมูลก็จะ

อตรงกับข้อมูลอินพุทและเอาท์พุทของระบบ ส่วนสัญญาณที่จะควบคุมหน่วยความจำ

7. การอ้างแอดเดรสแบบใช้รีจิสเตอร์ (register addressing) การออกแบบกลุ่มคำในขอบเขตของจำนวนบิทออปโค้ดจำกัดต้องหาวิธีที่ให้ได้ประสิทธิภาพดีที่สุด วิธีหนึ่งที่ใช้คือ การกำหนดรหัสของรีจิสเตอร์ เช่น มีวิธีรีจิสเตอร์ 8 ตัว ก็ใช้รหัส 3 บิทดังนั้น กลุ่มคำสั่งโหลดสามารถใช้คำสั่งเนื้อง 1 ไบท์ (8 บิท) ทำการโหลดข้อมูลระหว่างรีจิสเตอร์ได้เช่น

```
          LD          C          B
01        rd                     rs
```

8. การอ้างแอดเดรสแบบ implied addressing ในกรณีซีพียู จะตีความหมายเองโดยตรงว่า รีจิสเตอร์ภายในซีพียูตัวหนึ่งจะเป็นโอเปอร์แรนด์ เช่น คำสั่ง

```
          ADD                    Ar
                    10001                r
```

จากคำสั่งนี้จะเห็นว่า เราไม่ต้องกำหนดรหัสรีจิสเตอร์เลยแต่เครื่องจะตีความแล้วทราบเองว่าเป็นคำสั่งที่ต้องการกระทำร่วมกับรีจิสเตอร์ A

9. การอ้างแอดเดรสเข้าสู่บิทต่าง ๆ (bit addressing)ภายในตัว Z-80 มีคำสั่งนิเศษ ที่สามารถกระทำการเซทรีเซท หรือการทดสอบบิทใดบิทหนึ่งใน 8 บิทได้เช่น

```
RES       b,      r,      (reset  bit   b   of   orerand r)

                  CB

01                b                    r
```

ในที่นี้เราใช้รหัส b แทนโอเปอร์แลนด์ ว่าเป็นบิทใดในรีจิสเตอร์ ถ้า b= 000 หมายถึง 0 และ b = 111 หมายถึง บิท 7 ดังนั้นหมายถึงทำให้บิท 0 ของรีจิสเตอร์ A เป็น 0

กลุ่มคำสั่ง

กลุ่มคำสั่งของ Z-80 มีถึง 158 คำสั่ง โดยคำสั่งส่วนหนึ่งเป็นคำสั่งที่มีใน 8080 (78 คำสั่ง) คำสั่งเหล่านี้แบ่งออกเป็นกลุ่ม ๆ ได้ดังนี้

1. กลุ่มคำสั่งเกี่ยวกับการโหลดข้อมูลหรือแลกเปลี่ยนที่อยู่ข้อมูล
2. กลุ่มคำสั่งในการค้นหาข้อมูล และเคลื่อนย้ายข้อมูลเป็นกลุ่ม
3. กลุ่มคำสั่งในการกระทำทางคณิตศาสตร์และลอจิก
4. กลุ่มคำสั่งในการเลื่อนข้อมูลเป็นวงรอบและชิฟท์
5. กลุ่มคำสั่งในการทดสอบเซทและรีเซทบิทต่าง ๆ ของข้อมูล
6. กลุ่มคำสั่งในการอ้างถึงโปรแกรมย่อย
7. กลุ่มคำสั่งเกี่ยวข้องกับอินพุท เอาท์พุท
8. กลุ่มคำสั่งควบคุมการทำงานของ ซีพียู

กลุ่มคำสั่งเกี่ยวกับโหลดข้อมูลและแลกเปลี่ยนข้อมูล

การโหลดเป็นการเคลื่อนย้ายข้อมูลที่สำคัญ ใน Z-80 นั้นสามารถเคลื่อนย้ายข้อมูลได้อย่างมีประสิทธิภาพดีเพื่อทำความเข้าใจกลุ่มคำสั่งโหลดเรานิจารณาตัวอย่างต่าง ๆ ดังนี้

```
LD      E,      (IX + 08)
```

ชุดคำสั่งนี้ในหน่วยความจำจะเป็น

แอดเดรส	A	DD	
	A+1	5E	ออปโค้ด
	A+2	08	

จะเห็นว่าเป็นคำสั่งขนาดเพียง 3 ไบท์เท่านั้น และจากคำสั่งขนาด 3 ไบท์ เราสามารถใช้วิธีการเพิ่มขนาดของแอดเดรส (extended addressing) ได้เช่น

LD A, (6F 32)

ชุดคำสั่งจะเป็นแอดเดรส

A	3A	ออปโค้ด
A + 1	32	แอดเดรสนัยสำคัญต่ำ
A + 2	6F	แอดเดรสนัยสำคัญสูง

ลักษณะคำสั่งข้างบน จะเป็นการโหลดข้อมูลในหน่วยความจำตำแหน่ง แอดเดรส 6F 32 สิ่งที่น่าสังเกตคือ 32 จะอยู่ในไบท์ที่ 2 ส่วน 6F จะอยู่ในไบท์ที่ 3

เรายังโหลดข้อมูลในหน่วยความจำที่มีความสามารถสูงวิธีหนึ่งคือ การโหลดใช้อินเดอร์รีจิสเตอร์เป็นตัวรับข้อมูล 18 บิท ได้อีกด้วย และเพื่อลดขนาดของคำสั่งให้สั้นขึ้น ตัวคำสั่งจึงเป็นลักษณะของการแลกเปลี่ยนข้อมูลของคู่รีจิสเตอร์ต่าง ๆ

<u>กลุ่มคำสั่งในการค้นหาและเคลื่อนย้ายข้อมูลเป็นกลุ่ม</u>

Z-80 มีคำสั่งที่ทำให้เป็นไปอย่างมีประสิทธิภาพโดยทำให้ลดขนาดโดยทำให้ลดขนาดตัวโปรแกรมลงได้มาก มีลักษณะไวของกลุ่มคำสั่งในกลุ่มนี้จะอาศัยการทำงานรวมกันของคู่ ภายใน CPU 3 คู่ คือ

HL เป็นคู่รีจิสเตอร์ที่อ้างตำแหน่งจุดต้นทาง

DE เป็นคู่รีจิสเตอร์ที่อ้างถึงตำแหน่งจุดปลายทาง

BC เป็นตัวนับจำนวนไบท์

ในการค้นหาข้อมูล เราใช้วิธีการดังนี้เช่น

CPI (compare with increment)

ลักษณะคำสั่งจัดได้ดังนี้

แอดเดรส	A	EO ออปโค๊ด
	A + 1	A1

การทำงานจะเป็นไปดังนี้ CPU จะนำข้อมูลจากหน่วยความจำที่มีแอดเดรสค่าอยู่ในคู่รีจิสเตอร์ HL มาเปรียบเทียบกับรีจิสเตอร์ A หลังจากนี้ค่าตัวเองคู่รีจิสเตอร์ HL จะเพิ่มค่าอีก 1 และค่าของคู่รีจิสเตอร์ BC จะลดค่าตัวเองลงไปทีละ 1 ลักษณะการเปรียบเทียบจะให้ผลลัพท์ที่แฟลกจากคำสั่งนี้เราจะให้การเซทค่า BC เป็นจำนวนไบท์ที่ต้องการเปรียบ ค่าของ BC จะลดลงมาจนเป็น 0 แล้วเราทดสอบค่า BC =0 ได้เห็นว่าทุกครั้งที่ทำคำสั่งนี้ค่าคู่รีจิสเตอร์ HL จะเพิ่มค่า ดังนั้นถ้าให้กระทำคำสั่งนี้ใหม่แอดเดรสในหน่วยความจำจะเพิ่มค่า ดังนั้นถ้าให้กระทำคำสั่งที่สามารถให้ CPU กระทำคำสั่งจนครบตามจำนวนไบท์ ที่วางไว้ เช่น

CPIR (Compare with increment and repeat)

การจัดวางคำสั่งในหน่วยความจำจะเป็น

แอดเดรส		ออปโค๊ด
A	ED	
A + 1	B1	

การทำคำสั่งนี้จะเหมือนคำสั่ง CPI แต่ CPU จะเพิ่มการทดสอบค่าในคู่รีจิสเตอร์ BC ว่าเป็น 0 แล้วหรือยัง ถ้ายังก็จะกระทำสั่งเดิมซ้ำอีก ซึ่งค่าใน BC จะลดลงมาทีละ 1 จนเป็น 0 จึงหยุดกระทำ

นอกจากนี้เรายังสามารถให้ค่าแอดเดรส ในคู่รีจิสเตอร์ HL ลดลงทีละ 1 ได้เช่นกัน โดยใช้คำสั่ง CPD หรือ CPDR

สำหรับการเคลื่อนย้ายข้อมูลเป็นบล๊อก ก็กระทำเช่นเดียวการเปรียบเทียบโดยใช้คู่รีจิสเตอร์ HL เป็นตัวบอกแอดเดรสของหน่วยความจำที่จะนำไปเก็บอยู่ที่ใด ดังตัวอย่าง

LDI load location (DE) with location (HL)
increment
LDI (DE) (HL)

การทำรูปคำสั่งในหน่วยความจำจะเป็นดังนี้

		ออปโค๊ด
A	ED	
A + 1	A0	

การทำงานในลักษณะนี้ CPU นำข้อมูลในหน่วยความจำที่แอดเดรส (HL) ไปเก็บไว้ที่หน่วย
ความจำที่แอดเดรส (DE) และหลังจากนั้นค่าในคู่รีจิสเตอร์ (BC) และค่าในคู่รีจิสเตอร์ HL และ DE
จะเพิ่มค่าขึ้น 1 จะลดลงไป 1 ในทำนองเดียวกันเราสามารถใช้ลำพังเนื่องคำสั่งเดียวในการเคลื่อน
ย้ายข้อมูลทั้งบล็อกได้ด้วยการเอาจำนวนข้อมูลไว้ในคู่รีจิสเตอร์ BC แล้วใช้คำสั่ง LDIR การทำงาน
ของคำสั่งนี้จะคล้ายกับ LDI แต่จะทำซ้ำ ๆ ไปเรื่อยๆ จนกระทั่งค่าในคู่รีจิสเตอร์ BC เป็น 0 ก็จะ
หยุดไปกระทำคำสั่งถัดไป

<u>กลุ่มคำสั่งการกระทำทางคณิตศาสตร์และลอจิก</u>

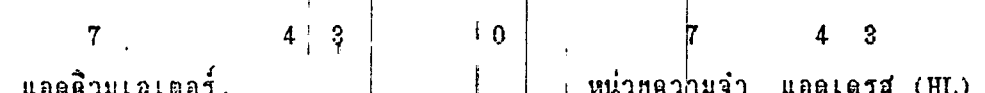
คำสั่งนี้จะกระทำด้วยรีจิสเตอร์ A เป็นส่วนใหญ่ลักษณะของกลุ่มคำสั่งประกอบด้วยลักษณะการ
ADD SUB, ADC, SBC, INC, DEC, AND, OR, XOR ในการอ้างแอดเดรสของตัวโอเปอร์แรนด์ นั้น
เราทำได้หลายแบบขึ้นอยู่กับผู้เขียนและผู้ใช้งาน

<u>กลุ่มคำสั่งในการเลื่อนข้อมูลเป็นวง (rotate) และการชิฟท์ (shift)</u>

ความสามารถพื้นฐานของ Z-80 ในการเลื่อนข้อมูลและชิฟท์ก็เหมือนกันใน CPU ของคอมพิว
เตอร์ทั่วไป แต่ใน Z 80 มีคำสั่งเกี่ยวกับการเลื่อนข้อมูลตัวเลข BCD อยู่ 2 คำสั่ง คือ RRD และ
RLD ทั้ง 2 คำสั่งนี้จะทำให้ข้อมูล 1 ตัวเลข (BCD) ในแอคคิวมูเลเตอร์เลื่อนเป็นวงรวมกับข้อมูล 2
ตัวเลข BCD ในหน่วยความจำที่อ้างแอดเดรสโดย HL ลักษณะการกระทำเป็นดังนี้

คำสั่ง RRD

คำสั่ง RLD



กลุ่มคำสั่งเกี่ยวกับการกระทำภายในข้อมูลบิท ต่าง ๆ

ใน Z 80 มีความหมายพิเศษในการเซท รีเซท หรือ ทดสอบบิทใดบิทหนึ่งในรีจิสเตอร์ต่าง ๆ และในหน่วยความจำได้รายละเอียดของการเซท , รีเซท และการทดสอบสามารถแยกเป็นคำสั่งย่อย ๆ ได้ถึง 8 คำสั่ง

กลุ่มคำสั่งที่อ้างกับโปรแกรมย่อย

เพื่อให้การเรียกโปรแกรมย่อยมีประสิทธิภาพสูง Z-80 ใช้วิธีการเรียกโดยคำสั่ง CALL และ RET คำสั่ง CAL จะกระทำได้ค่า PC เดิมที่มีอยู่ไปเก็บไว้ที่แสตคแล้วโหลดค่าแอดเดรสของกันนั่นเองจะเป็นการ POP เอาข้อมูลในสแตคมาให้ PC

นอกจากนี้ Z 80 มีคำสั่งพิเศษคือ

CALL cc, nn

11 cc 100

n

n

คำสั่งนี้เป็นการทดสอบเงื่อนไขใน cc ถ้าเงื่อนไขเป็นจริงกระทำ CALL ก็จะเกิดขึ้น เครื่องจะ PUSH ข้อมูลจริงการกระทำถัดไป การใช้รหัสแทนเงื่อนไข 3 บิท ทำให้สามารถแทนเงื่อนไขได้กับ 8 แบบ ดังนี้

cc	เงื่อนไข	ดูจากแฟลก
000	ไม่เป็นศูนย์	Z
001	เป็นศูนย์	Z
010	ไม่มีตัวทด	C
011	มีตัวทด	C
100	พาริตี้ เป็นคี่	P
101	พาริตี้ เป็นคู่	P
110	มีเครื่องหมายบวก	S

กลุ่มคำสั่งเกี่ยวกับการติดต่อระหว่างอินพุท- เอาท์พุทกับซีพียู ใน 8080 กลุ่มคำสั่งนี้มีเพียง 2 คำสั่งเท่านั้นคือ IN และ OUT แต่ในของ Z-80 ได้เพิ่มขยายการติดต่อกับอินพุท - เอาท์พุทมากยิ่งขึ้นทำให้ประสิทธิภาพการติดต่อเป็นไปได้ยิ่งขึ้น

คำสั่งที่ซ้ำกับ 8080 คือ

IN A PORT

ลักษณะการจัดรูปโครงสร้างคำสั่งในหน่วยความจำเป็น

แอดเดรส	A	DB
		ออปโค๊ด
	A ! 1	PORT

การกระทำคำสั่งนี้ จะเป็นการนำข้อมูลจากพอร์ทที่เป็นอินพุทโหลดข้อมูลมาที่รีจิสเตอร์ A สิ่งที่เพิ่มมาของ Z-80 คือ มันสามารถเลือกการโหลดให้กับข้อมูลไว้ที่รีจิสเตอร์ใดก็ได้เช่น

IN r, (C)

การจัดรูปคำสั่งในหน่วยความจำเป็น

A	ED
	ออปโค๊ด
A ! 1	01 ! r ! 000

ในกรณีซีพียู จะใช้ รีจิสเตอร์ C เป็นตัวกำหนดชื่อพอรทแล้วโหลดข้อมูลจากอินพุทมาเก็บไว้ที่รีจิสเตอร์ในซีพียูที่กำหนดด้วย 3 บิทในออปโค๊ด

การโหลดข้อมูลจาก I/O สามารถทำเป็นบล๊อกได้เช่นกันโดยใช้คำสั่ง INI, INIR, INI

A	ED
	ออปโค๊ด
A ! 1	A2

ในกรณีที่จะใช้รีจิสเตอร์ C เป็นตัวกำหนดชื่อพอร์ทและใช้รีจิสเตอร์ B เป็นตัวนำข้อมูลโดยค่า

ของรีจิสเตอร์ B จะลดลงมา 1 ส่วน รีจิสเตอร์ HL จะเป็นตัวชี้ตำแหน่งในหน่วยความจำที่จะนำข้อมูลจากอินพุทมาเก็บไว้ จะเห็นว่าถ้าเราให้ทำคำสั่งนี้ ซ้ำ ๆ ค่ารีจิสเตอร์ B จะลดลงมาเรื่อย ๆ ทำให้การเก็บข้อมูลเรียงกันไปได้

<u>คำสั่งเกี่ยวกับการควบคุมการทำงานของ</u> CPU มีหลายคำสั่งเช่น NOP ซึ่งเป็นคำสั่งที่ CPU ไม่ต้องทำอะไรเลย HALT เป็นคำสั่งให้หยุด CPU นอกจากนี้มีคำสั่งเกี่ยวกับการอินเตอร์รัพท์ ก็เช่นเดียวกัน (DI, EI, IM0, IM1 ,IM2)

จะพิจารณาการทำงานของคำสั่งเกี่ยวกับการอินเตอร์รัพท์ ต่าง ๆ เหล่านี้

DI	EI
F3	FB
ก	ข

ก. ความหมายของคำสั่งนี้จะกระทำการรีเซท ค่า IFF1 และ IFF2 ให้เป็น 0 ทั้งคู่ เนื่องแสดงลักษณะของการดิสเอเบิ้ล ดังนั้นการอินเตอร์รัพท์แบบมาสเคเบิ้ล ไม่ว่าแบบใดจะไม่สามารถได้รับผลตอบแทน

ข. เป็นคำสั่งไบท์เดียว ทำให้ IFF1 และ IFF2 เป็น 1 แต่การกระทำคำสั่งนี้เนื่องอันเดียวไม่สามารถอนาเบิ้ล การอินเตอร์รัพท์จะได้ต่อเมื่อ CPU ทำคำสั่งถัดไปเสร็จก่อน เพราะในคำสั่งที่ตามหลังอาจใช้เป็นคำสั่ง RET1 ซึ่งกรณีนี้จะต้องทำ RET1 ให้เสร็จก่อน

<u>ผลตอบแทนต่อนอนมาสเคเบิ้ลอินเตอร์รัพท์</u>

ถ้าสัญญาณ NMI เข้ามาพินจะตอบสนองทันทีโดยจะกระโดดไปกระทำยังตำแหน่งแอดเดรส 0066 ทันทีโดยค่าของ PC เดิมจะเก็บไว้บนสแตค ถ้าต้องการให้กลับมาทำในโปรแกรมหลัก เมื่อเสร็จการอินเตอร์รัพท์แล้ว ผู้โปรแกรมจะใช้คำสั่ง RETN (Return from nonmaskable interrupt) ค่าในสแตคจะได้รับการ POP มาไว้ใน PC เพื่อกระทำกลับไปยังโปรแกรมหลัก

<u>การอินเตอร์รัพท์โหมด 0</u>

ลักษณะการอินเตอร์รัพท์ในโหมดนี้จะเหมือนกับของ 8080 นั่นเองคือ เมื่อ CPU อยู่ในโหมดนี้ เมื่อเนอริเฟอรัลนี้ไปตีความหมายในบัสข้อมูล CPU จะเอาคำสั่ง 1 ไบท์ ที่ได้จากเนอริเฟอร์รัลนี้ไปความหมายใน IR (instruction register) คำสั่งนี้จะแทรกเข้าไปก็คือคำสั่ง 1 ไบท์ ที่ได้จากใช้ RST b นั่นเอง ดังนั้นเมื่อ CPU กระทำ RST b ข้อมูลของ PC จะเก็บไว้ในที่สแตค โดยPC ใหม่จะเก็บค่า b ที่มาจาก RST แล้วกระทำคำสั่งถัดไป ตามโปรแกรมเคาน์เตอร์

<u>อาวินเตอร์รันต์ โหมด 1</u>

ในซีพียู 8080 จะไม่มี จักษะของคำสั่งที่เซทรูปแบบของการอินเตอร์รันท์จะเป็นดังนี้

FB

ED

3C

โหมดนี้ CPU จะให้ผลตอบสนองต่อสัญญาณอินเตอร์รันท์โดยการทำการเอ็กซีคิวคำสั่ง RST 0038 โดยอัตโนมัติ นั่นคือถ้า CPU ในโหมดนี้ และมีการอินเตอร์รันท์เกิดขึ้น มันจะกระโดดไปกระทำที่ตำแหน่ง 0038 โดยอัตโนมัติ

FB

CF

อาวินเตอร์รันต์โหมดนี้จะเป็นแบบที่ประสิทธิภาพสูงสุดเพราะจะทำให้ CPU สามารถกระโดดไปยังตำแหน่งใดของโปรแกรมก็ได้ การกระทำในกรณีนี้จะใช้วิธีการเรียกแอดเดรสโดยทางอ้อมไปยังหน่วยความจำเป็นที่ใด ๆ โดยใช้ข้อมูลขนาด 8 บิท ที่ได้มาจากเนอริเปอรัล ซึ่งจะเป็นส่วนของบิทที่เป็นนัยสำคัญต่ำ รวมกับ I ที่เป็นบิทที่มีนัยสำคัญสูง โดยความจริงแล้วบิทนัยสำคัญต่ำสุดคือ A0 จะต้องเป็น 0 จะชี้ไปยังตำแหน่งในหน่วยความจำแล้วนำเอาค่าในหน่วยความจำนั้น 2 ไบท์ เรียงกันเป็นแอดเดรส ของกระโดดไปทำงาน

ซีพียูและคำสั่งของ Z-80

Z-80 มีมากกว่า 8080 ถึงเท่าตัวและยืดความสามารถของคำสั่งหลายอย่างที่สูงกว่ามาก ในที่นี้เราจะสรุปลักษะของคำสั่งเป็นตารางซึ่งแยกตามกลุ่มของคำสั่งได้ดังต่อไปนี้

สรุปคำสั่งของไมโครโปรเซสเซอร์เบอร์ Z-80

*Address Bus: A0 - A7: [C]
A8 - A15: [B]

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
I/O	IN	A,PORT	2		[A]←[PORT] Input to Accumulator from directly addressed I/O port. Address bus: A0-A7: PORT A8-A15: [A]
	IN	REG,(C)	2	X X P X 0	[REG]←[[C]] Input to register from I/O port addressed by the contents of C.* If second byte is 70_{16} only the flags will be affected.
	INR		2	1 ? ? ? 1	Repeat until [B]=0: [[HL]]←[[C]] [B]←[B]-1 [HL]←[HL]+1 Transfer a block of data from I/O port addressed by contents of C to memory location addressed by contents of HL, going from low addresses to high. Contents of B serve as a count of bytes remaining to be transferred.*
	INDR		2	1 ? ? ? 1	Repeat until [B]=0. [[HL]]←[[C]] [B]←[B]-1 [HL]←[HL]-1 Transfer a block of data from I/O port addressed by contents of C to memory location addressed by contents of HL, going from high addresses to low. Contents of B serve as a count of bytes remaining to be transferred.*
	INI		2	X ? ? ? 1	[[HL]]←[[C]] [B]←[B]-1 [HL]←[HL]+1 Transfer a byte of data from I/O port addressed by contents of C to memory location addressed by contents of HL. Decrement byte count and increment destination address.*

INSTRUCTION		OBJECT CODE	BYTES	CLOCK PERIODS	8080A MNEMONIC		8080A CLOCK PERIODS
RRC	REG	CB 00001xxx	2	8			
RRCA		0F	1	4	RRC		4
RRD		ED 67	2	18			
RST	N	11xxx111	1	11	RST	N	11
SBC	DATA	DE YY	2	7	SBI	DATA	7
SBC	(HL)	9E	1	7	SBB	M	7
SBC	HL,RP	ED 01xx0010	2	15			
SBC	(IX + DISP)	DD 9E YY	3	19			
SBC	(IY + DISP)	FD 9E YY	3	19			
SBC	REG	10011xxx	1	4	SBB	REG	4
SCF		37	1	4	STC		4
SET	B,(HL)	CB 11bbb110	2	15			
SET	B,(IX + DISP)	DD CB YY 11bbb110	4	23			
SET	B,(IY + DISP)	FD CB YY 11bbb110	4	23			
SET	B,REG	CB 11bbbxxx	2	8			
SLA	(HL)	CB 26	2	15			
SLA	(IX + DISP)	DD CB YY 26	4	23			
SLA	(IY + DISP)	FD CB YY 26	4	23			
SLA	REG	CB 00100xxx	2	8			
SRA	(HL)	CB 2E	2	15			
SRA	(IX + DISP)	DD CB YY 2E	4	23			
SRA	(IY + DISP)	FD CB YY 2E	4	23			
SRA	REG	CB 00101xxx	2	8			
SRL	(HL)	CB 3E	2	15			
SRL	(IX + DISP)	DD CB YY 3E	4	23			
SRL	(IY + DISP)	FD CB YY 3E	4	23			
SRL	REG	CB 00111xxx	2	8			
SUB	DATA	D6 YY	2	7	SUI	DATA	7
SUB	(HL)	96	1	7	SUB	M	7
SUB	(IX + DISP)	DD 96 YY	3	19			
SUB	(IY + DISP)	FD 96 YY	3	19			
SUB	REG	10010xxx	1	4	SUB	REG	4
XOR	DATA	EE YY	2	7	XRI	DATA	7
XOR	(HL)	AE	1	7	XRA	M	7
XOR	(IX + DISP)	DD AE YY	3	19			
XOR	(IY + DISP)	FD AE YY	3	19			
XOR	REG	10101xxx	1	4	XRA	REG	4

x represents an optional binary digit.
bbb represents optional binary digits identifying a bit location in a register or memory byte.
ddd represents optional binary digits identifying a destination register.
sss represents optional binary digits identifying a source register.
pppp represents a four hexadecimal digit memory address.
YY represents two hexadecimal data digits.
YYYY represents four hexadecimal data digits.

When two possible execution times are shown (i.e., 5/11), it indicates that the number of clock periods depends on condition flags.

Execution time shown is for one iteration.

สรุปรหัสคำสั่งที่อยู่ในรูปนีโมนิค รหัสภาษาเครื่องและช่วงเวลาในการประมวลผล

INSTRUCTION	OBJECT CODE	BYTES	CLOCK PERIODS	8080A MNEMONIC	8080A CLOCK PERIODS
OUT PORT,A	D3 YY	2	11	OUT PORT	10
OUTD	ED AB	2	15		
OUTDR	ED BB	2	20/15		
OUTI	ED A3	2	15		
OUTIR	ED B3	2	20/15		
POP IX	DD E1	2	14		
POP IY	FD E1	2	14		
POP PR	11xx0001	1	10	POP RP	10
PUSH IX	DD E5	2	15		
PUSH IY	FD E5	2	15		
PUSH PR	11xx0101	1	11	PUSH RP	11
RES B,(HL)	CB 10bbb110	2	15		
RES B,(IX + DISP)	DD CB YY 10bbb110	4	23		
RES B,(IY + DISP)	FD CB YY 10bbb110	4	23		
RES B,REG	CB 10bbbxxx	2	8		
RET	C9	1	10	RET	10
RET C	D8	1	5/11	RC	5/11
RET M	F8	1	5/11	RM	5/11
RET NC	D0	1	5/11	RNC	5/11
RET NZ	C0	1		RNZ	5/11
RET P	F0	1	5/11	RP	5/11
RET PE	E8	1	5/11	RPE	5/11
RET PO	E0	1	5/11	RPO	5/11
RET Z	C8	1	5/11	RZ	5/11
RETI	ED 4D	2	14		
RETN	ED 45	2	14		
RL (HL)	CB 16	2	15		
RL (IX + DISP)	DD CB YY 16	4	23		
RL (IY + DISP)	FD CB YY 16	4	23		
RL REG	CB 00010xxx	2	8		
RLA	17	1	4	RAL	4
RLC (HL)	CB 06	2	15		
RLC (IX + DISP)	DD CB YY 06	4	23		
RLC (IY + DISP)	FD CB YY 06	4	23		
RLC REG	CB 00000xxx	2	8		
RLCA	07	1	4	RLC	4
RLD	ED 6F	2	18		
RR (HL)	CB 1E	2	15		
RR (IX + DISP)	DD CB YY 1E	4	23		
RR (IY + DISP)	FD CB YY 1E	4	23		
RR REG	CB 00011xxx	2	8		
RRA	1F	1	4	RAR	4
RRC (HL)	CB 0E	2	15		
RRC (IX + DISP)	DD CB YY 0E	4	23		
RRC (IY + DISP)	FD CB YY 0E	4	23		

สรุปรหัสคำสั่งที่อยู่ในรูปนีโมนิค รหัสภาษาเครื่องและช่วงเวลาในการประมวลผล

INSTRUCTION	OBJECT CODE	BYTES	CLOCK PERIODS	8080A MNEMONIC	8080A CLOCK PERIODS
LD (ADDR),HL	22 pppp	3	16	SHLD ADDR	16
LD (ADDR),IX	DD 22 pppp	4	20		
LD (ADDR),IY	FD 22 pppp	4	20		
LD (ADDR),SP	ED 73 pppp	4	20		
LD (BC),A	02	1	7	STAX B	7
LD (DE),A	12	1	7	STAX D	7
LD HL,(ADDR)	2A pppp	3	16	LHLD ADDR	16
LD (HL),DATA	36 YY	2	10	MVI M,DATA	10
LD (HL),REG	01110sss	1	7	MOV M,REG	7
LD I,A	ED 47	2	9		
LD IX,(ADDR)	DD 2A pppp	4	20		
LD IX,DATA16	DD 21 YYYY	4	14		
LD (IX + DISP),DATA	DD 36 YY YY	4	19		
LD (IX + DISP),REG	DD 01110sss YY	3	19		
LD IY,(ADDR)	FD 2A pppp	4	20		
LD IY,DATA16	FD 21 YYYY	4	14		
LD (IY + DISP),DATA	FD 36 YYYY	4	19		
LD (IY + DISP),REG	FD 01110sss YY	3	19		
LD R,A	ED 4F	2	9		
LD REG,DATA	00ddd110 YY	2	7	MVI REG,DATA	7
LD REG,(HL)	01ddd110	1	7	MOV REG,M	7
LD REG,(IX + DISP)	DD 01ddd110 YY	3	19		
LD REG,(IY + DISP)	FD 01ddd110 YY	3	19		
LD REG,REG	01dddsss	1	4	MOV REG,REG	5
LD RP,(ADDR)	ED 01xx1011 pppp	4	20		
LD RP,DATA16	00xx0001 YYYY	3	10	LXI RP,DATA16	10
LD SP,HL	F9	1	6	SPHL	5
LD SP,IX	DD F9	2	10		
LD SP,IY	FD F9	2	10		
LDD	ED A8	2	16		
LDDR	ED B8	2	21/16		
LDI	ED A0	2	16		
LDIR	ED B0	2	21/16		
NEG	ED 44	2	8		
NOP	00	1	4	NOP	4
OR DATA	F6 YY	2	7	ORI DATA	7
OR (HL)	B6	1	7	ORA M	7
OR (IX + DISP)	DD B6 YY	3	19		
OR (IY + DISP)	FD B6 YY	3	19		
OR REG	10110xxx	1	4	ORA REG	5
OUT (C),REG	ED 01sss001	2	12		

สรุปรหัสคำสั่งที่อยู่ในรูปนีโมนิค รหัสภาษาเครื่องและช่วงเวลาในการประมวลผล

INSTRUCTION		OBJECT CODE	BYTES	CLOCK PERIODS	8080A MNEMONIC		8080A CLOCK PERIODS
DEC	REG	00xxx101	1	4	DCR	REG	5
DI		F3	1	4	DI		4
DJNZ	DISP	10 YY	2	8/13			
EI		FB	1	4	EI		4
EX	AF,AF	08	1	4			
EX	DE,HL	EB	1	4	XCHG		4
EX	(SP),HL	E3	1	19	XTHL		18
EX	(SP),IX	DD E3	2	23			
EX	(SP),IY	FD E3	2	23			
EXX		D9	1	4			
HALT		76	1	4	HLT		4
IM	0	ED 46	2	8			
IM	1	ED 56	2	8			
IM	2	ED 5E	2	8			
IN	A,PORT	DB YY	2	10	IN	PORT	10
IN	REG,(C)	ED	2	11			
		01ddd000					
INC	(HL)	34	1	11	IMR	M	10
INC	IX	DD 23	2	10			
INC	(IX + DISP)	DD 34 YY	3	23			
INC	IY	FD 23	2	10			
INC	(IY + DISP)	FD 34 YY	3	23			
INC	RP	00xx0011	1	6	INX	RP	5
INC	REG	00xxx100	1	4	INR	REG	5
IND		ED AA	2	15			
INDR		ED BA	2	20/15			
INI		ED A2	2	15			
INIR		ED B2	2	20/15			
JP	LABEL	C3 ppqq	3	10	JMP	LABEL	10
JP	C,LABEL	DA ppqq	3	10	JC	LABEL	10
JP	(HL)	E9	1	4	PCHL		5
JP	(IX)	DD E9	2	8			
JP	(IY)	FD E9	2	8			
JP	M,LABEL	FA ppqq	3	10	JM	LABEL	10
JP	NC,LABEL	D2 ppqq	3	10	JNC	LABEL	10
JP	NZ,LABEL	C2 ppqq	3	10	JNZ	LABEL	10
JP	P LABEL	F2 ppqq	3	10	JP	LABEL	10
JP	PE LABEL	EA ppqq	3	10	JPE	LABEL	10
JP	PO,LABEL	E2 ppqq	3	10	JPO	LABEL	10
JP	Z,LABEL	CA ppqq	3	10	JZ	LABEL	10
JR	C,DISP	38 YY	2	7/12			
JR	DISP	18 YY	2	12			
JR	NC,DISP	30 YY	2	7/12			
JR	NZ,DISP	20 YY	2	7/12			
JR	Z,DISP	28 YY	2	7/12			
LD	A,(ADDR)	3A ppqq	3	13	LDA	ADDR	13
LD	A,(BC)	0A	1	7	LDAX	B	7
LD	A,(DE)	1A	1	7	LDAX	D	7
LD	A,I	ED 57	2	9			
LD	A,R	ED 5F	2	9			
LD	(ADDR),A	32 ppqq	3	13	STA	ADDR	13
LD	(ADDR),BC	ED 43 ppqq	4	20			
LD	(ADDR),DE	ED 53 ppqq	4	20			

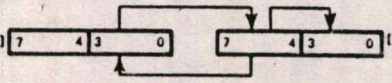
สรุปรหัสคำสั่งที่อยู่ในรูปนีโมนิค รหัสภาษาเครื่องและช่วงเวลาในการประมวลผล

INSTRUCTION		OBJECT CODE	BYTES	CLOCK PERIODS	8080A MNEMONIC		8080A CLOCK PERIODS
ADC	DATA	CE YY	2	7	ACI	DATA	7
ADC	(HL)	8E	1	7	ADC	M	7
ADC	HL,RP	ED 01xx1010	2	15			
ADC	(IX + DISP)	DD 8E YY	3	19			
ADC	(IY + DISP)	FD 8E YY	3	19			
ADC	REG	10001xxx	1	4	ADC	REG	4
ADD	DATA	C6 YY	2	7	ADI	DATA	7
ADD	(HL)	86	1	7	ADD	M	7
ADD	HL,RP	00xx1001	1	11	DAD	RB	10
ADD	(IX + DISP)	DD 86 YY	3	19			
ADD	IX,PP	DD 00xx1001	2	15			
ADD	(IY + DISP)	FD 86 YY	3	19			
ADD	IY,RR	FD 00xx1001	2	15			
ADD	REG	10000xxx	1	4	ADD	REG	4
AND	DATA	E6 YY	2	7	ANI	DATA	7
AND	(HL)	A6	1	7	ANA	M	7
AND	(IX + DISP)	DD A6 YY	3	19			
AND	(IY + DISP)	FD A6 YY	3	19			
AND	REG	10100xxx	1	4	ANA	REG	4
BIT	B,(HL)	CB 01bbb110	2	12			
BIT	B,(IX + DISP)	DD CB YY 01bbb110	4	20			
BIT	B,(IY + DISP)	FD CB YY 01bbb110	4	20			
BIT	B,REG	CB 01bbbxxx	2	8			
CALL	LABEL	CD ppqq	3	17	CALL	LABEL	17
CALL	C,LABEL	DC ppqq	3	10/17	CC	LABEL	11/17
CALL	M,LABEL	FC ppqq	3	10/17	CM	LABEL	11/17
CALL	NC,LABEL	D4 ppqq	3	10/17	CNC	LABEL	11/17
CALL	NZ,LABEL	C4 ppqq	3	10/17	CNZ	LABEL	11/17
CALL	P,LABEL	F4 ppqq	3	10/17	CP	LABEL	11/17
CALL	PE,LABEL	EC ppqq	3	10/17	CPE	LABEL	11/17
CALL	PO,LABEL	E4 ppqq	3	10/17	CPO	LABEL	11/17
CALL	Z,LABEL	CC ppqq	3	10/17	CZ	LABEL	11/17
CCF		3F	1	4	CMC		4
CP	DATA	FE YY	2	7	CPI	DATA	7
CP	(HL)	BE	1	7	CMP	M	7
CP	(IX + DISP)	DD BE YY	3	19			
CP	(IY + DISP)	FD BE YY	3	19	CMP	REG	19
CP	REG	10111xxx	1	4			
CPD		ED A9	2	16			
CPDR		ED B9	2	21/16			
CPI		ED A1	2	16			
CPIR		ED B1	2	21/16			
CPL		2F	1	4	CMA		4
DAA		27	1	4	DAA		4
DEC	(HL)	35	1	11	DCR	M	10
DEC	IX	DD 2B	2	10			
DEC	(IX + DISP)	DD 35 YY	3	23			
DEC	IY	FD 2B	2	10			
DEC	(IY + DISP)	FD 35 YY	3	23			
DEC	RP	00xx1011	1	6	DCX	RP	5

สรุปคำสั่งของไมโครโปรเซสเซอร์เบอร์ Z-80 (ต่อ)

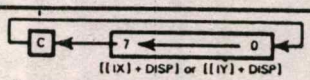
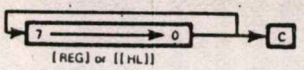
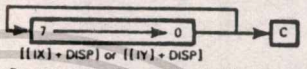
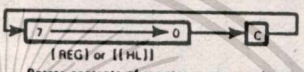
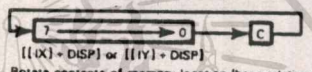
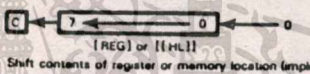
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
INTERRUPT	DI		1		Disable interrupts
	EI		1		Enable interrupts
	RST	N	1		PUSH PC, [PC]—$(8 \cdot N)_{16}$ Restart at designated location
	RETI		2		Return from interrupt
	RETN		2		Return from nonmaskable interrupt
	IM	0 1 2	2		Set interrupt mode 0, 1, or 2
STATUS	SCF		1	1 0 0	C—1 Set Carry flag
	CCF		1	X ? 0	C—$\overline{C}$ Complement Carry flag
	NOP		1		No operation — volatile memories are refreshed
	HALT		1		CPU halts, executes NOPs to refresh volatile memories.

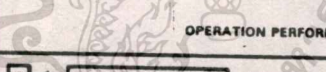
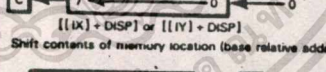
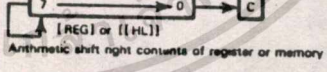
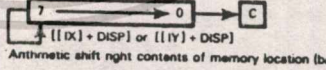
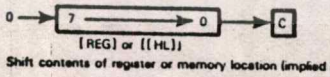
สรุปคำสั่งของไมโครโปรเซสเซอร์เบอร์ Z-80 (ต่อ)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
REGISTER OPERATE (Continued)	RRD		2	X X P 0 0	 Rotate one BCD digit right between the Accumulator and memory location (implied addressing). Contents of the upper half of the Accumulator are not affected.
BIT MANIPULATION	BIT	B,REG	2	X ? ? 1 0	Z—REG(B) Zero flag contains complement of the selected register bit
	BIT	B,(HL)	2	X ? ? 1 0	Z—[[HL]](B) Zero flag contains complement of selected bit of the memory location (implied addressing).
	BIT	B,(IX + DISP) B,(IY + DISP)	4	X ? ? 1 0	Z—[[IX] + DISP](B) or Z—[[IY] + DISP](B) Zero flag contains complement of the selected bit of the memory location (base relative addressing).
	SET	B,REG	2		REG(B)—1 Set indicated register bit
	SET	B,(HL)	2		[[HL]](B)—1 Set indicated bit of memory location (implied addressing).
	SET	B,(IX + DISP) B,(IY + DISP)	4		[[IX] + DISP](B)—1 or [[IY] + DISP](B)—1 Set indicated bit of memory location (base relative addressing).
	RES	B,REG	2		REG(B)—0 Reset indicated register bit
	RES	B,(HL)	2		[[HL]](B)—0 Reset indicated bit in memory location (implied addressing).
	RES	B,(IX + DISP) B,(IY + DISP)	4		[[IX] + DISP](B)—0 or [[IY] + DISP](B)—0 Reset indicated bit of memory location (base relative addressing).

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
STACK	PUSH	PR	1		[[SP]-1]—[PR(HI)] [[SP]-2]—[PR(LO)] [SP]—[SP]-2 Put contents of register pair on top of Stack and decrement Stack Pointer.
	PUSH	IX IY	2		[[SP]-1]—[IX(HI)] or [SP]-1]—[IY(HI)] [[SP]-2]—[IX(LO)] or [[SP]-2]—[IY(LO)] [SP]—[SP]-2 Put contents of index register on top of Stack and decrement Stack Pointer.
	POP	PR	1		[PR(LO)]—[[SP]] [PR(HI)]—[[SP] + 1] [SP]—[SP] + 2 Put contents of top of Stack in register pair and increment Stack Pointer.
	POP	IX IY	2		[IX(LO)]—[[SP]] or [IY(LO)]—[[SP]] [IX(HI)]—[[SP] + 1] or [IY(HI)]—[[SP] + 1] [SP]—[SP] + 2 Put contents of top of Stack in index register and increment Stack Pointer.
	EX	(SP),HL	1		[H]——[[SP] + 1] [L]——[[SP]] Exchange contents of HL and top of Stack.
	EX	(SP),IX (SP),IY	2		[IX(HI)]——[[SP] + 1] or [IY(HI)]——[[SP] + 1] [IX(LO)]——[[SP]] or [IY(LO)]——[[SP]] Exchange contents of index register and top of Stack.

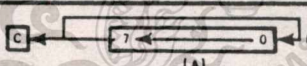
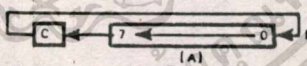
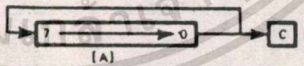
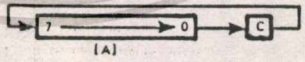
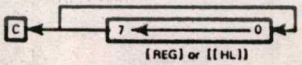
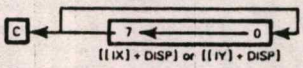
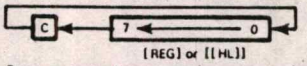
สรุปคำสั่งของไมโครโปรเซสเซอร์เบอร์ Z-80 (ต่อ)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
REGISTER OPERATE (Continued)	RL	(IX + DISP) (IY + DISP)	4	X X X P 0 0	 [(IX) + DISP] or [(IY) + DISP] Rotate contents of memory location (base relative addressing) left through Carry
	RRC	REG (HL)	2	X X X P 0 0	 [REG] or [(HL)] Rotate contents of register or memory location (implied addressing) right with branch Carry
	RRC	(IX + DISP) (IY + DISP)	4	X X X P 0 0	 [(IX) + DISP] or [(IY) + DISP] Rotate contents of memory location (base relative addressing) right with branch Carry
	RR	REG (HL)	2	X X X P 0 0	 [REG] or [(HL)] Rotate contents of register or memory location (implied addressing) right through Carry
	RR	(IX + DISP) (IY + DISP)	4	X X X P 0 0	 [(IX) + DISP] or [(IY) + DISP] Rotate contents of memory location (base relative addressing) right through Carry.
	SLA	REG (HL)	2	X X X P 0 0	 [REG] or [(HL)] Shift contents of register or memory location (implied addressing) left with branch Carry.

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
REGISTER OPERATE (Continued)	SLA	(IX + DISP) (IY + DISP)	4	X X X P 0 0	 [(IX) + DISP] or [(IY) + DISP] Shift contents of memory location (base relative addressing) left with branch Carry.
	SRA	REG (HL)	2	X X X P 0 0	 [REG] or [(HL)] Arithmetic shift right contents of register or memory location (implied addressing).
	SRA	[IX + DISP] [IY + DISP]	4	X X X P 0 0	 [(IX) + DISP] or [(IY) + DISP] Arithmetic shift right contents of memory location (base relative addressing).
	SRL	REG (HL)	2	X X X P 0 0	 [REG] or [(HL)] Shift contents of register or memory location (implied addressing) right with branch Carry
	SRL	(IX + DISP) (IY + DISP)	4	X X X P 0 0	 [(IX) + DISP] or [(IY) + DISP] Shift contents of memory location (base relative addressing) right with branch Carry.
	RLD		2	X X P 0 0	 Rotate one BCD digit left between the Accumulator and memory location (implied addressing). Contents of the upper half of the Accumulator are not affected.

สรุปคำสั่งของไมโครโปรเซสเซอร์เบอร์ Z-80 (ต่อ)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
REGISTER REGISTER OPERATE (Continued)	ADC	HL RP	2	X X X O ? 0	[HL] ← [HL] + [RP] + C 16-bit add with Carry; register pair contents to contents of HL
	SBC	HL RP	2	X X X O ? 1	[HL] ← [HL] - [RP] - C 16-bit subtract with Carry; register pair contents from contents of HL
	ADD	IX PP	2	X ? 0	[IX] ← [IX] + [PP] 16-bit add register pair contents to contents of Index register (PP BC, DE, IX, SP)
	ADD	IY RR	2	X ? 0	[IY] ← [IY] + [RR] 16-bit add register pair contents to contents of IY Index register (RR BC, DE, IY, SP)
REGISTER OPERATE	DAA		1	X X X P X	Decimal adjust Accumulator, assuming that Accumulator contents are the sum or difference of BCD operands.
	CPL		1	1 1	[A] ← - [A] Complement Accumulator (ones complement)
	NEG		2	X X X O X 1	[A] ← - [A] + 1 Negate Accumulator (twos complement)
	INC	REG	1	X X O X 0	[REG] ← [REG] + 1 Increment register contents
	INC	RP	1		[RP] ← [RP] + 1 Increment contents of register pair
	INC	IX IY	2		[IX] ← [IX] + 1 or [IY] ← [IY] + 1 Increment contents of Index register
	DEC	REG	1	X X O X 1	[REG] ← - [REG] - 1 Decrement register contents
	DEC	RP	1		[RP] ← - [RP] - 1 Decrement contents of register pair
	DEC	IX IY	2		[IX] ← [IX] - 1 or [IY] ← [IY] - 1 Decrement contents of Index register

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
REGISTER OPERATE (Continued)	RLCA		1	X 0 0	 Rotate Accumulator left with branch Carry
	RLA		1	X 0 0	 Rotate Accumulator left through Carry
	RRCA		1	X 0 0	 Rotate Accumulator right with branch Carry
	RRA		1	X 0 0	 Rotate Accumulator right through Carry
	RLC	REG (HL)	2	X X X P 0 0	 Rotate contents of register or memory location (implied addressing) left with branch Carry
	RLC	(IX + DISP) (IY + DISP)	4	X X X P 0 0	 Rotate contents of memory location (base relative addressing) left with branch Carry
	RL	REG (HL)	2	X X X P 0 0	 Rotate contents of register or memory location (implied addressing) left through Carry

สรุปคำสั่งของไมโครโปรเซสเซอร์เบอร์ Z-80 (ต่อ)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
JUMP ON CONDITION	JP	COND,LABEL	3		If COND, then [PC]—LABEL Jump to instruction at address LABEL if the condition is satisfied
	JR	C,DISP	2		If C=1, then [PC]—[PC]+2+DISP Jump relative to contents of Program Counter if Carry flag is set
	JR	NC,DISP	2		If C=0, then [PC]—[PC]+2+DISP Jump relative to contents of Program Counter if Carry flag is reset
	JR	Z,DISP	2		If Z=1, then [PC]—[PC]+2+DISP Jump relative to contents of Program Counter if Zero flag is set
	JR	NZ,DISP	2		If Z=0, then [PC]—[PC]+2+DISP Jump relative to contents of Program Counter if Zero flag is reset
	DJNZ	DISP	2		[B]—[B]-1 If [B]≠0, then [PC]—[PC]+2+DISP Decrement contents of B and Jump relative to contents of Program Counter if result is not 0
REGISTER-REGISTER MOVE	LD	DST,SRC	1		[DST]—[SRC] Move contents of source register to destination register. SRC and DST may each be A, B, C, D, E, H or L.
	LD	A,IV	2	X X I 0 0	[A]—[IV] Move contents of interrupt vector to Accumulator
	LD	A,R	2	X X I 0 0	[A]—[R] Move contents of Refresh register to Accumulator
	LD	IV,A	2		[IV]—[A] Load interrupt vector from Accumulator
	LD	R,A	2		[R]—[A] Load Refresh register from Accumulator
	LD	SP,HL	1		[SP]—[HL] Move contents of HL to Stack Pointer
	LD	SP,IX SP,IY	2		[SP]—[IX] or [SP]—[IY] Move contents of index register to Stack Pointer

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
REGISTER-REGISTER MOVE (Continued)	EX	DE,HL	1		[DE]⟷[HL] Exchange DE and HL contents
	EX	AF,AF'	1		[AF]⟷[AF'] Exchange program status and alternate program status
	EXX		1		$\begin{pmatrix}[BC]\\[DE]\\[HL]\end{pmatrix} \longleftrightarrow \begin{pmatrix}[BC']\\[DE']\\[HL']\end{pmatrix}$ Exchange register pairs and alternate register pairs.
REGISTER-REGISTER OPERATE	ADD	REG	1	X X X O X 0	[A]—[A]+[REG] Add contents of register to Accumulator
	ADC	REG	1	X X X O X 0	[A]—[A]+[REG]+C Add contents of register and Carry to Accumulator.
	SUB	REG	1	X X X O X 1	[A]—[A]-[REG] Subtract contents of register from Accumulator
	SBC	REG	1	X X X O X 1	[A]—[A]-[REG]-C Subtract contents of register and Carry from Accumulator
	AND	REG	1	0 X X P X 1	[A]—[A]∧[REG] AND contents of register with contents of Accumulator
	OR	REG	1	0 X X P X 0	[A]—[A]∨[REG] OR contents of register with contents of Accumulator
	XOR	REG	1	0 X X P X 0	[A]—[A]∨[REG] Exclusive-OR contents of register with contents of Accumulator
	CP	REG	1	X X X O X 1	[A] : [REG] Compare contents of register with contents of Accumulator
	ADD	HL,RP	1	X ? 0	[HL]—[HL]+[RP] 16-bit add register pair contents to contents of HL

สรุปคำสั่งของไมโครโปรเซสเซอร์เบอร์ Z-80 (ต่อ)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
SECONDARY MEMORY REFERENCE (Continued)	INC	(IX + DISP) (IY + DISP)	3	X X O X 0	[(IX) + DISP]←[(IX) + DISP] + 1 or [(IY) + DISP]←[(IY) + DISP] + 1 Increment using base relative addressing
	DEC	(HL)	1	X X O X 1	[(HL)]←[(HL)]-1 Decrement using implied addressing
	DEC	(IX + DISP) (IY + DISP)	3	X X O X 1	[(IX) + DISP]←[(IX) + DISP]-1 or [(IY) + DISP]←[(IY) + DISP]-1 Decrement using base relative addressing
IMMEDIATE	LD	REG,DATA	2		[REG]←DATA Load immediate into register
	LD	RP,DATA16	3		[RP]←DATA16 Load 16 bits of immediate data into register pair
	LD	IX,DATA16 IY,DATA16	4		[IX]←DATA16 or [IY]←DATA16 Load 16 bits of immediate data into index register
	LD	(HL),DATA	2		[(HL)]←DATA Load immediate into memory location using implied addressing
	LD	(IX + DISP),DATA (IY + DISP),DATA	4		[(IX) + DISP]←DATA or [(IY) + DISP]←DATA Load immediate into memory location using base relative addressing
JUMP	JP	LABEL	3		[PC]←LABEL Jump to instruction at address LABEL
	JR	DISP	2		[PC]←[PC) + 2 + DISP Jump relative to present contents of Program Counter
	JP	(HL)	1		[PC]←[HL] Jump to address contained in HL
	JP	(IX) (IY)	2		[PC]←[IX] or [PC]←[IY] Jump to address contained in index register

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
SUBROUTINE CALL AND RETURN	CALL	LABEL	3		[(SP)-1]←[PC(HI)] [(SP)-2]←[PC(LO)] [SP]←[SP]-2 [PC]←LABEL Jump to subroutine starting at LABEL
	CALL	COND,LABEL	3		Jump to subroutine if condition is satisfied; otherwise, continue in sequence
	RET		1		[PC(LO)]←[(SP)] [PC(HI)]←[(SP) + 1] [SP]←[SP] + 2 Return from subroutine
	RET	COND	1		Return from subroutine if condition is satisfied; otherwise, continue in sequence
IMMEDIATE OPERATE	ADD	DATA	2	X X X O X 0	[A]←[A] + DATA Add immediate to Accumulator
	ADC	DATA	2	X X X O X 0	[A]←[A] + DATA + C Add immediate with Carry
	SUB	DATA	2	X X X O X 1	[A]←[A]-DATA Subtract immediate from Accumulator
	SBC	DATA	2	X X X O X 1	[A]←[A]-DATA-C Subtract immediate with Carry
	AND	DATA	2	0 X X P X 1	[A]←[A]∧DATA AND immediate with Accumulator
	OR	DATA	2	0 X X P X 0	[A]←[A]∨ DATA OR immediate with Accumulator
	XOR	DATA	2	0 X X P X 0	[A]←[A]∀ DATA Exclusive-OR immediate with Accumulator
	CP	DATA	2	X X X O X 1	[A]: DATA Compare immediate data with Accumulator contents

สรุปคำสั่งของไมโครโปรเซสเซอร์เบอร์ Z-80 (ต่อ)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
BLOCK TRANSFER AND SEARCH (Continued)	CPI		2	X X X X 1	[A]:[[HL]] [HL]—[HL]+1 [BC]—[BC]-1 Compare contents of Accumulator with those of memory location addressed by contents of HL. Increment address and decrement byte count.
	CPD		2	X X X X 1	[A]:[[HL]] [HL]—[HL]-1 [BC]—[BC]-1 Compare contents of Accumulator with those of memory location addressed by contents of HL. Decrement address and byte count.
SECONDARY MEMORY REFERENCE	ADD	(HL)	1	X X X 0 X 0	[A]—[A]+[[HL]] Add to Accumulator using implied addressing.
	ADD	(IX + DISP) (IY + DISP)	3	X X X 0 X 0	[A]—[A]+[[IX]+DISP] or [A]—[A]+[[IY]+DISP] Add to Accumulator using base relative addressing
	ADC	(HL)	1	X X X 0 X 0	[A]—[A]+[[HL]]+C Add with Carry using implied addressing.
	ADC	(IX + DISP) (IY + DISP)	3	X X X 0 X 0	[A]—[A]+[[IX]+DISP]+C or [A]—[A]+[[IY]+DISP]+C Add with Carry using base relative addressing
	SUB	(HL)	1	X X X 0 X 1	[A]—[A]-[[HL]] Subtract from Accumulator using implied addressing
	SUB	(IX + DISP) (IY + DISP)	3	X X X 0 X 1	[A]—[A]-[[IX]+DISP] or [A]—[A]-[[IY]+DISP] Subtract using base relative addressing
	SBC	(HL)	1	X X X 0 X 1	[A]—[A]-[[HL]]-C Subtract with Carry using implied addressing

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
SECONDARY MEMORY REFERENCE (Continued)	SBC	(IX + DISP) (IY + DISP)	3	X X X 0 X 1	[A]—[A]-[[IX]+DISP]-C or [A]—[A]-[[IY]+DISP]-C Subtract with Carry using base relative addressing
	AND	(HL)	1	0 X X P X 1	[A]—[A]∧[[HL]] AND with Accumulator using implied addressing
	AND	(IX + DISP) (IY + DISP)	3	0 X X P X 1	[A]—[A]∧[[IX]+DISP] or [A]—[A]∧[[IY]+DISP] AND with Accumulator using base relative addressing
	OR	(HL)	1	0 X X P X 0	[A]—[A]∨[[HL]] OR with Accumulator using implied addressing
	OR	(IX + DISP) (IY + DISP)	3	0 X X P X 0	[A]—[A]∨[[IX]+DISP] or [A]—[A]∨[[IY]+DISP] OR with Accumulator using base relative addressing
	XOR	(HL)	1	0 X X P X 0	[A]—[A]⊻[[HL]] Exclusive-OR with Accumulator using implied addressing
	XOR	(IX + DISP) (IY + DISP)	3	0 X X P X 0	[A]—[A]⊻[[X]+DISP] or [A]—[A]⊻[[IY]+DISP] Exclusive-OR with Accumulator using base relative addressing
	CP	(HL)	1	0 X X O X 1	[A]:[[HL]] Compare with Accumulator using implied addressing
	CP	(IX + DISP) (IY + DISP)	3	0 X X O X 1	[A]:[[IX]+DISP] or [A]:[[IY]+DISP] Compare with Accumulator using base relative addressing
	INC	(HL)	1	X X O X 0	[[HL]]—[[HL]]+1 Increment using implied addressing

สรุปคำสั่งของไมโครโปรเซสเซอร์เบอร์ Z-80 (ต่อ)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
PRIMARY MEMORY REFERENCE (Continued)	LD	REG,(HL)	1		(REG)←((HL)) Load register from memory location addressed by contents of HL.
	LD	(BC),A (DE),A	1		((BC))←(A) or ((DE))←(A) Store Accumulator to memory location addressed by the contents of the specified register pair.
	LD	(HL),REG	1		((HL))←(REG) Store register contents to memory location addressed by the contents of HL.
	LD	REG,(IX + DISP) REG,(IY + DISP)	3		(REG)←((IX) + DISP) or (REG)←((IY) + DISP) Load register from memory location using base relative addressing.
	LD	(IX + DISP),REG (IY + DISP),REG	3		((IX) + DISP)←(REG) or ((IY) + DISP)←(REG) Store register to memory location addressed relative to contents of index register.
BLOCK TRANSFER AND SEARCH	LDIR		2	0 0 0	Repeat until (BC)=0: ((DE))←((HL)) (DE)←(DE) + 1 (HL)←(HL) + 1 (BC)←(BC)-1 Transfer a block of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE, going from low addresses to high. Contents of BC serve as a count of bytes to be transferred.
	LDDR		2	0 0 0	Repeat until (BC)=0: ((DE))←((HL)) (DE)←(DE)-1 (HL)←(HL)-1 (BC)←(BC)-1 Transfer a block of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE, going from high addresses to low. Contents of BC serve as a count of bytes to be transferred.

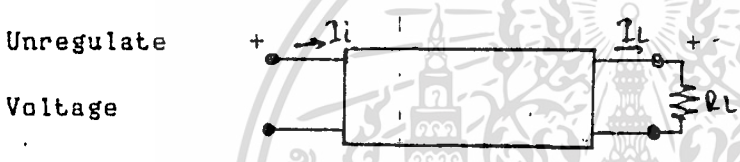
TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES C Z S P/O A_C N	OPERATION PERFORMED
BLOCK TRANSFER AND SEARCH (Continued)	LDI		2	X 0 0	((DE))←((HL)) (DE)←(DE) + 1 (HL)←(HL) + 1 (BC)←(BC)-1 Transfer one byte of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE. Increment source and destination addresses and decrement byte count.
	LDD		2	X 0 0	((DE))←((HL)) (DE)←(DE)-1 (HL)←(HL)-1 (BC)←(BC)-1 Transfer one byte of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE. Decrement source and destination addresses and byte count.
	CPIR		2	X X X X 1	Repeat until (A)=((HL)) or (BC)=0: (A):((HL)) (HL)←(HL) + 1 (BC)←(BC)-1 Compare contents of Accumulator with those of memory block addressed by contents of HL, going from low addresses to high. Stop when a match is found or when the byte count becomes zero.
	CPDR		2	X X X X 1	Repeat until (A)=((HL)) or (BC)=0: (A):((HL)) (HL)←(HL)-1 (BC)←(BC)-1 Compare contents of Accumulator with those of memory block addressed by contents of HL, going from high addresses to low. Stop when a match is found or when the byte count becomes zero.

บทที่ 4

วงจรจ่ายไฟตรงรักษาระดับแรงดัน (ELECTRONIC VOLTAGE REGULATOR)

เครื่องใช้อิเลคทรอนิคส์จำนวนมาก ต้องใช้แรงดันรักษาระดับโดยแรงดันที่เครื่องใช้ได้รับต้องคงที่สม่ำเสมอไม่ว่าจะมีการเปลี่ยนแปลงของแรงดันจ่ายเข้า กระแสโหลดหรืออุณหภูมิ หากจะเขียนความสัมพันธ์ของแรงดันจ่ายออกซึ่งรักษาระดับกับแรงดันจ่ายเข้า กระแสโหลด และอุณหภูมิเราได้ว่า

$$dv_O = \frac{\partial v_O}{\partial v_i} dv_i + \frac{\partial v_O}{\partial I_L} dI_L + \frac{\partial v_O}{\partial T} dT$$



รูปที่ 4.1

ถ้า 1. Stabilization Factor $SV = \frac{\partial V_O}{\partial v_i}$ เมื่อ IL กับ T คงที่และ dv_i

หมายถึงความเปลี่ยนแปลงของแรงดันจ่ายเข้า

2. Output Resistance ของวงจรรักษาระดับแรงดัน

$$R_O = -\frac{\partial V_O}{\partial I_L}$$ เมื่อ VI กับ T คงที่

หมายเหตุ เทอม RO มีเครื่องหมายเป็น ลบเนื่องจาก VO ลดลง เมื่อ IL เพิ่มขึ้นและเนื่องจาก IL ไหลออกจาก Regulator unit

3. Temperature Co-Efficient $ST = \frac{\partial V_O}{\partial T}$ เมื่อ Vi และ IL คงที่จะได้ว่า

$$\partial V_O = S_V dv_i - R_O dI_L + S_T dT$$

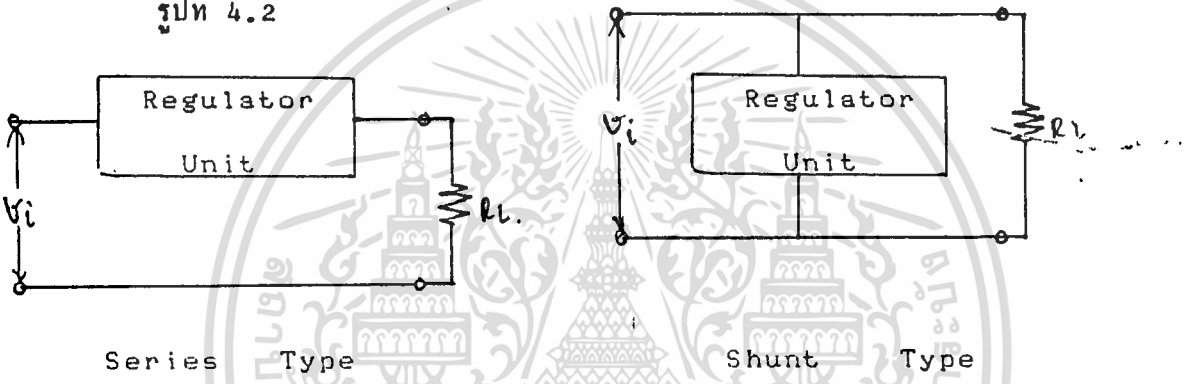
โครงสร้างวงจรรักษาระดับแรงดัน

วงจรรักษาระดับแรงดันสามารถสร้างขึ้นจากอุปกรณ์อิเลคทรอนิคส์ที่มีคุณ Dynamic-Resistance ต่ำ มีแรงดันจ่ายออก ที่เราต้องการ ซึ่งอุปกรณ์ที่ว่านี้ก็คือ Zener Diode หรือวงจรที่เกิดจากการนำเอา Zener Diode ไปต่อรวมอยู่ก็ได้

วงจรรักษาแรงดันสามารถจำแนกประเภทตามสถาพการต่อโหลดได้ 2 แบบ คือ

1. แบบขนาน (Shunt Type)
2. แบบอนุกรม (Series Type)

รูปที่ 4.2



Series Type Shunt Type

ข้อแตกต่างที่สำคัญของวงจรรักษาระดับแรงดันสองแบบนี้สามารถสรุปได้ดังนี้

1. ประสิทธิภาพของวงจรแบบอนุกรมจะพิจารณาที่แรงดัน เนื่องมาจากกระแส input กับกระแส output จะมีค่าใกล้เคียงกัน $\eta = \frac{vo}{Vi}$

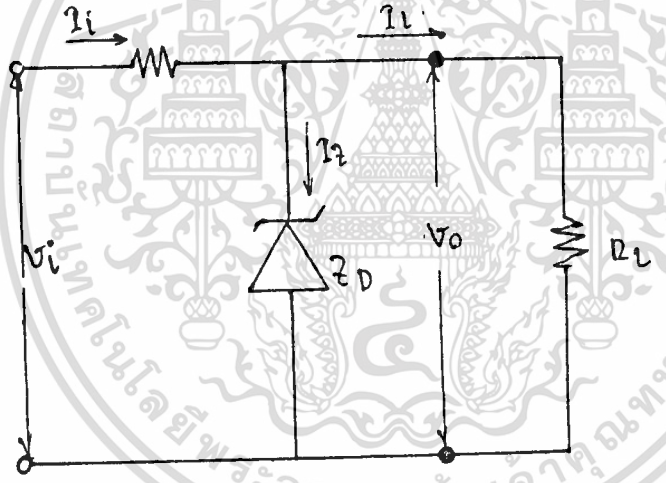
2. ประสิทธิภาพของวงจรแบบขนานจะพิจารณาที่กระแส นี้เนื่องมาจากแรงดัน input กับแรง output จะมีค่าใกล้เคียงกัน $\eta = \frac{Io}{Ii}$

3. วงจรแบบขนานมักใช้งานเมื่อ Vo และ RL คงที่ นอกจากนี้ยังนิยมใช้เฉพาะในกรณีที่แรงดันจ่ายออกมีค่าต่ำ กระแสจ่ายเข้ามีค่าสูง

4. วงจรแบบขนานจะมีราคาต่ำกว่าแบบอนุกรม เนื่องมาจากเป็นวงจรแบบง่าย
5. วงจรแบบขนาน ไม่จำเป็นต้องมีการป้องกันกระแสไหลเกิน
6. วงจรอนุกรมนิยมใช้งานกับงานโดยทั่วไป

ในบทนี้จะได้นำเอาวงจรรักษาระดับแรงดันแบบต่างๆ มาวิเคราะห์พอเป็นพื้นฐานแก่การศึกษาในระดับสูงขึ้นและสามารถนำไปประยุกต์ใช้งานได้

วงจรรักษาระดับแรงดันแบบซีเนอร์ไดโอด



รูปที่ 4.1

เป็นวงจรรักษาระดับ แรงดันพื้นฐานซึ่งประยุกต์ใช้คุณสมบัติของ ซีเนอร์ไดโอดจากรูปวงจร โดยใช้กฎแรงเคลื่อนของ เคอร์ชอฟฟ์

$$Vi = (Iz + IL)\,Rd + Vo$$

โดยใช้กฎของโอหม์

$$I_L = \frac{V_O}{R_L} \quad , \quad I_Z = \frac{V_O}{R_Z}$$

เมื่อ Rz = ความต้านทานของ ซีเนอร์ไดโอด

$$V_i = \left(\frac{V_O}{R_Z} + \frac{V_O}{R_L} \right) R_D + V_O$$

$$= V_O \left(\frac{R_D}{R_Z} + \frac{R_D}{R_1} + 1 \right)$$

$$\frac{V_I}{V_O} = \frac{K_D}{R_Z} + \frac{R_D}{R_L} + 1$$

$$\frac{V_O}{V_1} = \frac{1}{1 + \frac{R_D}{R_Z} + \frac{R_D}{R_1}}$$

$$S_V = \frac{\partial V_O}{\partial V_I} \simeq \frac{V_{Oo}}{V_I}$$

$$S_V = \frac{1}{1 + \frac{D_R}{R_Z} + \frac{D_R}{R_1}}$$

ถ้า $R_L \gg R_D \gg R_Z$

$$S_V = \frac{R_Z}{R_D}$$

จากรูปวงจร RO = RZ // RD

วงจรรักษาระดับแรงดันแบบนี้มี Temperature Co-efficient ต่ำ นอกจากนี้ยังมีการสูญเสียกำลังใน ซีเนอร์ไดโอดเป็นจำนวนมาก จึงมีประสิทธิภาพค่อนข้างต่ำ

ในการคำนวณหาพิกัดของอุปกรณ์ในวงจรสามารถคำนวณได้โดยใช้สูตร

$$VZ = VO$$

$$PZ\ (MAX) = \left[\frac{VI - VO - IL\ (MIN)}{RD}\right] VZ$$

$$Rd\ (MAX) = \frac{VI(MIN) - VZ\ (MAX)}{IL\ (MIN) + IL\ (MAX)}$$

$$RD\ (MIN) = \frac{VI\ (MAX) - VZ\ (MIN)}{\frac{PZ\ (MAX)}{VZ\ (MAX)} + IL\ (MIN)}$$

<u>ตัวอย่าง</u> วงจรรักษาระดับแรงดันแบบ ซีเนอร์ไดโอด ดังรูปในบทเรียนมีค่า Rd = 200 Ohm., Vz =20 Ohm. แรงดันจ่ายออกที่มีค่าปกติเท่ากับ 10 V. ที่กระแสปกติ 5 mA จงคำนวนหา

1. การเปลี่ยนแปลงของแรงดันจ่ายออกเมื่อแรงดันจ่ายเข้าเปลี่ยนแปลงไป 1.1 V. และกระแสโหลดคงที่ 5 mA

2. การเปลี่ยนแปลงของแรงดันที่จ่ายออกเมื่อแรงดันจ่ายเข้าคงที่ และกระแสโหลดเปลี่ยนแปลงจาก 5 mA ไปสู่ 1 mA

<u>Solu</u> จากโจทย์เราจะได้ว่า $RL = \frac{VO}{IL} = 2{,}000$ Ohm.

1. จากสมาการ $SV = \dfrac{1}{1 + \frac{RD}{RZ} + \frac{RD}{RL}} = \dfrac{1}{1 + \frac{200}{20} + \frac{200}{2,000}}$

$$= \frac{1}{11.1}$$

จาก $$SV = \frac{\Delta VO}{\Delta VI}$$

$$\Delta VO = SV \, \Delta VI = \frac{1}{11.1} .1.1 = 0.099 \text{ V.} \quad \underline{\text{Ans}}$$

2. จาก $$RO = RZ \,//\, RD = \frac{20}{20 + 200} \times \frac{200}{} = \frac{200}{11}$$

$$RO = \frac{\partial VO}{\partial IL} \simeq \frac{\Delta VO}{\Delta IL}$$

$$\therefore \Delta VO = \Delta IL \cdot RO$$

$$= (5-1) \times 10^{-3} \times \frac{200}{11} = 0.073 \quad \underline{\text{Ans}}$$

<u>ตัวอย่าง</u> จงคำนวณหาค่า Rd สำหรับวงจรรักษาระดับแรงดันแบบ ซีเนอร์ไดโอด ดังรูป ในบทเรียนกำหนด Vi (max) = 15 V. Vo (max) = 7.5 V. IL (max =100 mA. Vi (min) = 13.5 V. Vz (min) = 7.3 V. Iz (min) = 20 mA. IL (min) = 8.2 V. และ Vi (max) = 16.5 V.

$$\text{SO/ จาก} \quad RD(MAX) = \frac{VI(MIN) - VZ(MAX)}{IZ(MIN) + IL(MAX)}$$

$$= \frac{13.5 - 8.2}{20\times10^{-3} - 100\times10^{3}} = 44.17 \text{ Ohm.}$$

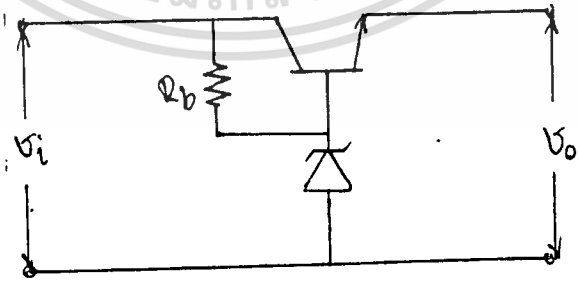
$$RD(MIN) = \frac{VI(MAX) - VZ(MIN)}{\frac{PZ}{VZ(MAX)} + IL(MIN)} = 40.8 \text{ Ohm.}$$

$$RD = \frac{RD(MAX) + RD(MIN)}{2}$$

$$= 42.48 \text{ Ohm.} \quad \underline{\text{Ans}}$$

Single Bjt Series Regulator

เรียกอีกอย่างหนึ่งว่า Emitter Follower Regulator เป็นวงจรที่ปรับปรุงข้อบกพร่องของแบบ ซีเนอร์ เพื่อให้วงจรใช้งานกับกระแสที่มากขึ้นและมี Sv ดีขึ้น วงจรแบบนี้จัดได้ว่าเป็นแบบ Series Regulator



รูปที่ 4.4

$$SV = \frac{\partial VO}{\partial V.I} = \frac{V\rho}{vi} = \frac{VZ}{Rd + RZ}$$

$$RO = \frac{HIE + RZ}{HIE + 1}$$

$$P_d = (VI\ mAX) - VO\ (MIN)\ \ IL\ (MAX)$$

$$I_B\ (MAX) = \frac{IL\ (MAX)}{HFE + 1}$$

$$RD\ (MAX) = \frac{VI\ (MIN) - VZ\ (MAX)}{IB\ (MAX) + IZ\ (MAX)}$$

$$RD\ (MIN) = \frac{VI\ (MAX) - VZ\ (MIN)}{\frac{PZ}{VZ\ (MIN)} + IB\ (MIN)}$$

$$RD = \frac{RD\ (MAX) + RD\ (MIN)}{2}$$

ตัวอย่าง จากวงจร Emitter Follower Regulator ถ้า Vi = 18 +- 2 V, Vo (in) = 12 V, IL 50 ถึง 250 mA จงคำนวนหา Pd (max) ของไดโอด Sv, Ro ของวงจรและค่า Rd ของวงจรและค่า Rd เมื่อ ทรานซิสเตอร์มีค่า hfe = 44 และ hie = 5.5 Ohm. ที่ IC = 250 mA, Vz =12 C Vz (max) 12.6 V, Vz (min) = 11.4 V, Pz (max) = 500 mw, hfe (max) =130, Vz (max) =11.7 V, Iz (min) = 5.44 mA

SO/N PD (MAX) = (VI(MAX) - VO (MIN) IL (MAX)

= (20 -12) x $250x10^{-3}$ = 2 WAtt

$$IB\ (MAX) = \frac{IL\ (MAX)}{HFE + 1} = \frac{250x10^{-3}}{44 + 1} = 5.56\ mA$$

$$RD\ (MAX) = \frac{VI\ (MIN) - VZ\ (MAX)}{IB\ (MAX) + IZ\ (MIN)}$$

$$= \frac{16 - 12.6}{(5.56 + 5.44)x\ 10^{-3}} = 309\ Ohm.$$

$$IB\ (MIN) = \frac{IL\ (MIN)}{HFE\ (MAX)} = \frac{50x10^{-3}}{130} = 0.385\ mA$$

$$RD\ (MIN) = \frac{VI\ (MAX) - VZ\ (MIN)}{\frac{PZ}{VZ\ (MIN)} + IB\ (MIN)}$$

$$= \frac{20 - 11.4}{\frac{0.5}{11.4} + 0.385x10^{-3}} = 194.4\ Ohm$$

$$RD = \frac{RD\ (MAX) + RD\ (MIN)}{} = 251.8$$

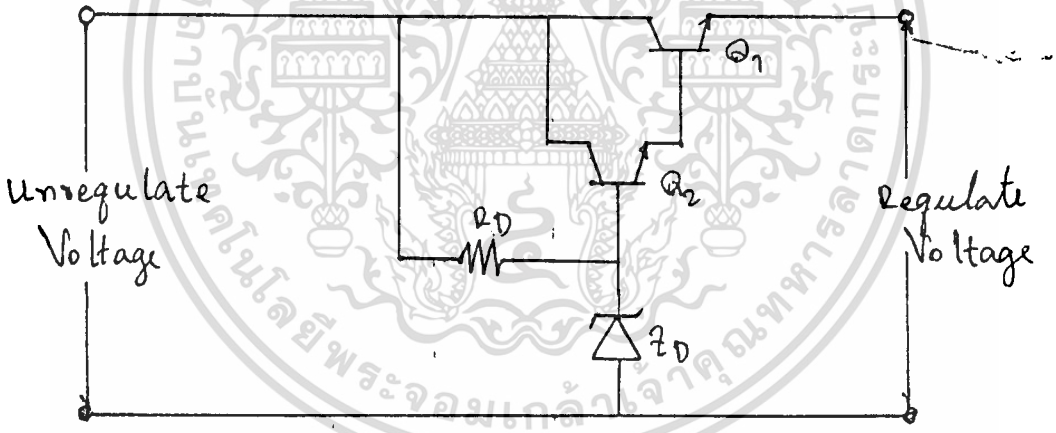
$$RO = \frac{RZ + HIE}{HFE + 1} = \frac{12 + 5.5}{44 + 1}$$

= 0.388 Ohm.

$$SV = \frac{RZ}{FD + RZ}$$

$$= \frac{12}{251.8 + 12} = 0.0445$$

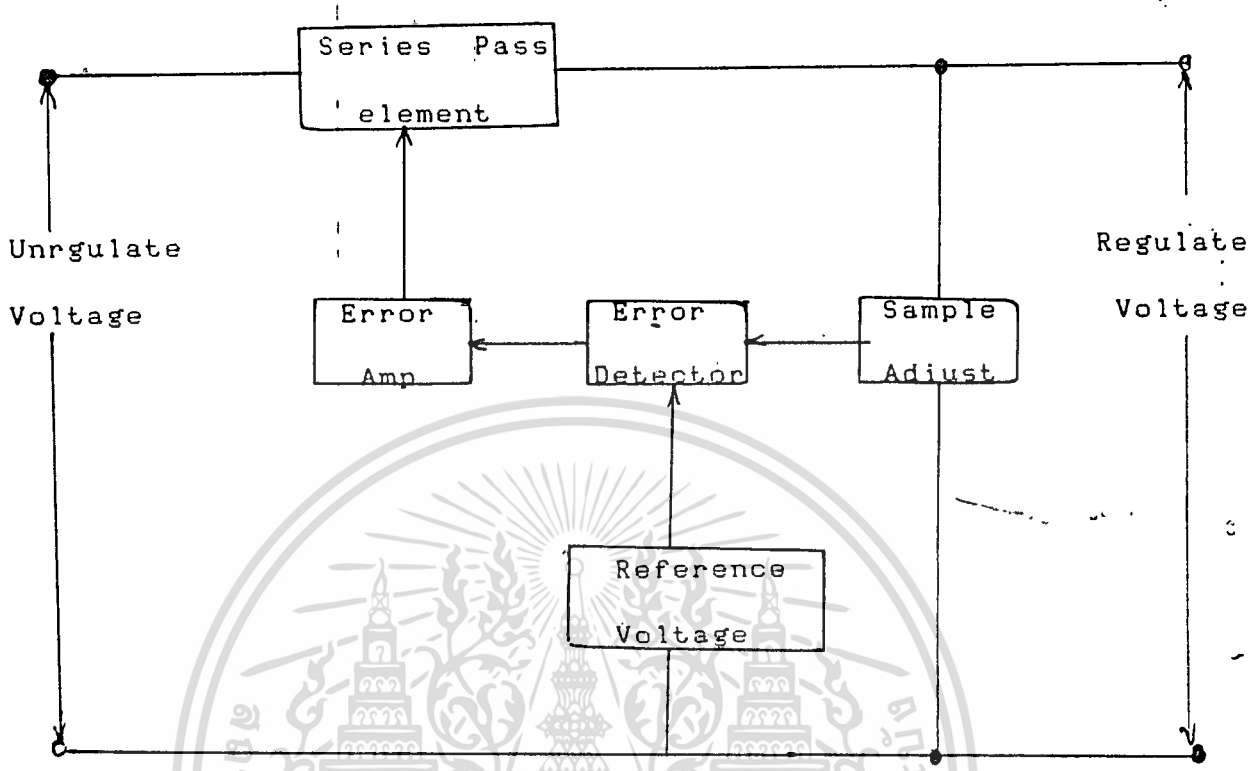
ข้อเสียของวงจรแบบนี้คือ ในกรณีที่กระแสโหลดสูงเกินไปค่า Pd ของ Zener จะมีค่ามาก ซึ่งทำให้วงจรมีราคาแพงแต่อาจแก้ไขได้โดยใช้ทรานซิสเตอร์ที่มีค่า hfe สูง หรือใช้ทรานซิสเตอร์ที่มีค่า hfe สูง หรือใช้ทรานซิสเตอร์ที่มี hfe ต่ำ 2 ตัว ต่อแบบ Dartington ดังรูป



รูปที่ 4.5

Feed Back Regulato

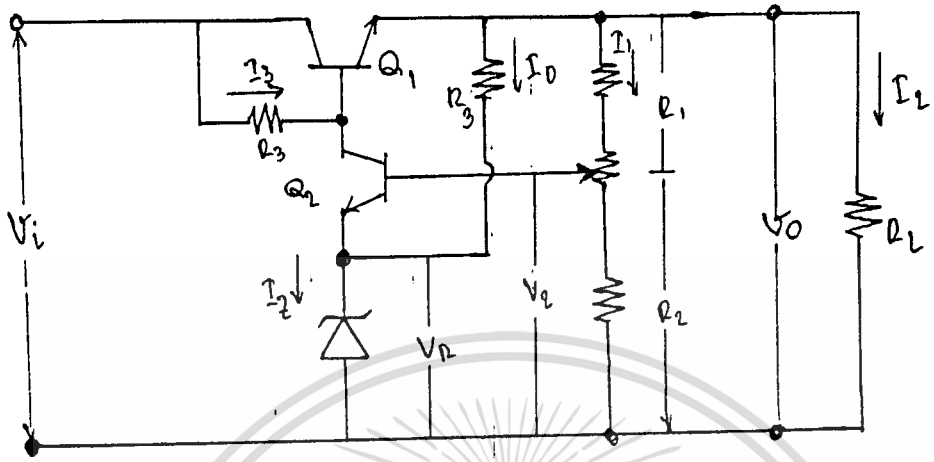
เป็นแบบที่นิยมใช้งานกันมากที่สุดแบบหนึ่ง เพราะเป็นวงจรที่มีเสถียรภาพทางแรงดันดีมาก เพื่อความสะอาดในการอธิบายการทำงานของวงจร Feed Back Regulator จะใช้ Block Diagram ข้างล่างนี้ประกอบ



รูปที่ 4.6

วงจร Sampling ซึ่งต่อขนานไว้กับ output จะทำการสุ่มตัวอย่างโดยการใช้วงจร โวลเตอดิ ไวเตอร์ นำเอาแรงดันที่จ่ายออก output บางส่วนมาตรวจสอบเปรียบเทียบกับแรงดันมาตรฐาน (Reference Voltage) ว่าแตกต่างกันเท่าใดในภาค Error-Detector หากมีความแตกต่างกัน เพียงเล็กน้อยก็จะถูกขยายให้มีความแรงมากขึ้นในภาค Error Amp สัญญานจากภาค Error Amp จะไปควบคุมความต้านทานของ Series pass-Element ให้มีค่าความต้านทานที่เหมาะสม

จากรูปเป็นวงจร Transistor Feed Back Regulator อย่างง่าย



รูปที่ 4.7

ในการออกแบบวงจรดังกล่าวสามารถทำได้โดยทำตามลำดับดังนี้

1. $IZ = ID2 + IE2$

2. $RD = \frac{VO - VR}{ID}$

3. $IB2 = \frac{IC2}{HFE2}$

4. $V2 = VB2 + VR$

5. $R1 = \frac{VO - V2}{I1}$

6. $R2 = \frac{V2}{I1}$

7. $IB1 = \frac{IL + IL + ID}{HFE1} = \frac{IO}{HFE1} = \frac{IC}{HFE1}$

8. $I3 = IB1 + IC2$

9. $R3 = \frac{VI - (VBE1 + VO)}{I3}$

นอกจากนี้เรายังสามารถคำนวนหาคุณสมบัติของวงจรได้โดยใช้สูตร

$$SV = \frac{1}{GM \; R3}$$

$$GM = HFE2 \cdot \frac{R2}{R1+R2} \cdot \frac{1}{(R1//E2)+HIE2 + (1+HFE2)\,RZ}$$

$$RO = \frac{RO + (R3 + HIE1) / (C1 + HFE1)}{1 + GM\,(R3 + 80)}$$

$$VO = +- \; SV \; \Delta \; VI + RO \; \Delta \; IL$$

<u>ตัวอย่าง</u> จงออกแบบ Feed Back Regulator กำหนด

$VO = 25$ V

$IL \leq 1$ Amp.

$VI = 50 +- 5$ V ที่ ro = 10 Ohm.

แล้วคำนวนหาค่า VO เมื่อ IL เปลี่ยนจาก 0 - 1 Amp และ VI เปลี่ยนแปลงไป

I 5 E

SO/^{2} เลือก VR - <u>VO</u> โดยใช้ zenner diode ซึ่งมี vz =7.5v 2 ตัวเมื่อ

จ่าย เป็น VR = 15 ซึ่งจะทำให้เกิด RZ = 12 ที่ IZ = 20 MA

2 ตัวต่ออันดับกันเพื่อจ่าย เป็น VR = 15 V ซึ่งจะทำให้เกิด rz = 12 Ohm. ที่ IZ = 20 mA

* หมายเหตุ ข้อมูล Iz และ Iz หาได้จากคู่มือ Zener Diode

ประมาณให้ IC2 - IE2 = 10 mA คือทนได้ ถึง 30 mA ที่ VCE (max) = 45 V

คู่มือทรานซิสเตอร์ถ้า ICL = 10 mA

HFE = 220, HFC = 200, HIE = 800 Ohm.

เลือก ID = 10 mA

1. IZ = IC2 + IE2 = 20 mA

2. $RD = \frac{VO - VR}{ID} = 1\ K\ Ohm.$

3. $IB2 = \frac{IC2}{HFE2} = 4\ \mu A$

เราต้องการให้ I1 >> IB2 เลือก I1 = 10 mA ที่ VBE = 0.7 V

4. V2 = VBE2 + VR = 15.7 V

5. $R1 = \frac{VO - VZ}{I1} = 930\ Ohm.$

6. $R2 \simeq \frac{VZ}{IL} = 1,570\ Ohm.$

เลือก ZN 1722 Silicon Power Transistor เป็น Q1 ที่ IC = 1 mA

hFE = 125, hfe1 = 100, hie = 20 Ohm.

7. IB1 = IL + I1 + ID / HFE1 - 8 mA

8. I3 = IB1 + IC2 = 18 mA

9. $R3 = \frac{Vi - (VBE1 + VO)}{I} = 1,350$

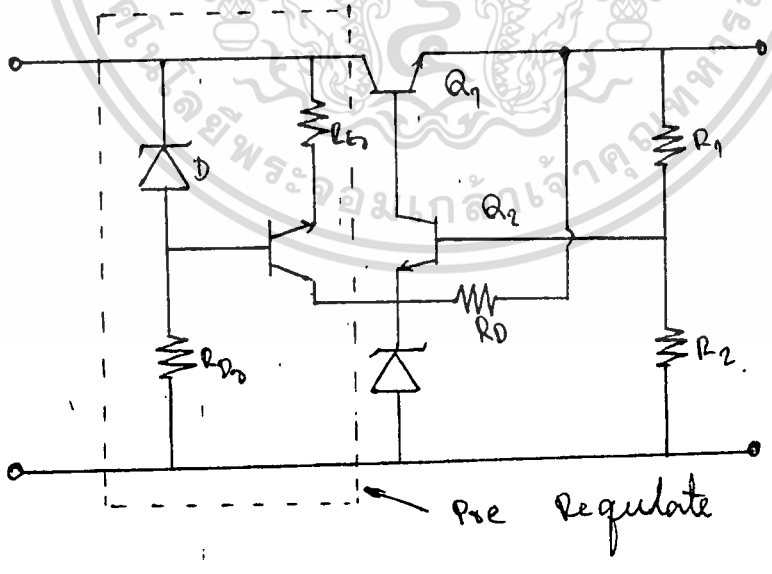
SV = 0.022

GM = $\frac{1}{SV \quad R3}$ = 0.033, Ro = 0.51 Ohm, Vo = 0.73

Ans

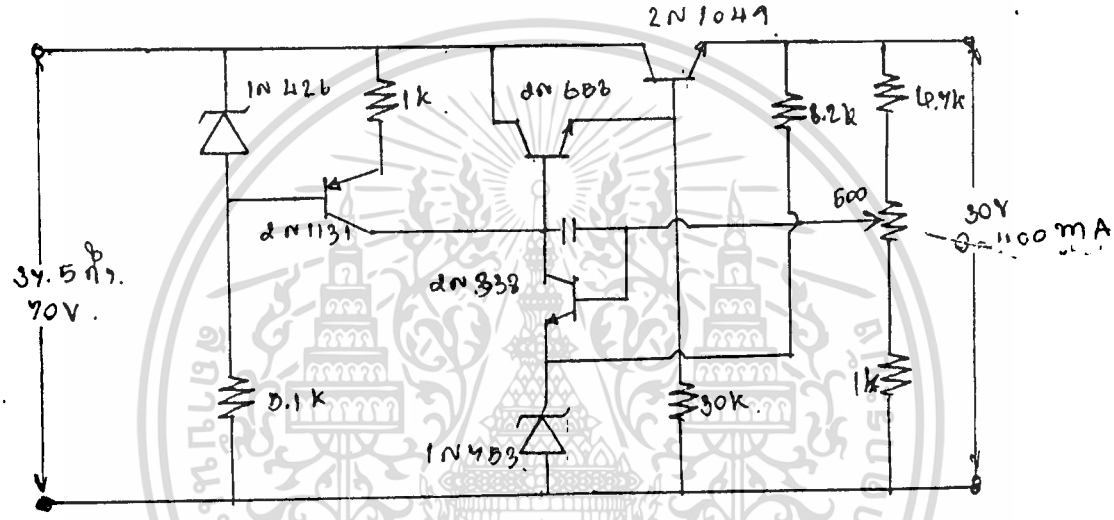
High Performance Feed Buck Regulator

ในกรณีต้องการ ให้วงจร Feed Back Regulator ทำงานได้ดีและมีประสิทธิภาพสูงขึ้นอาจปรับปรุงโดยแทนที่ตัวต้านทาน R3 ด้วย ภาค Perl Regulator (ซึ่งเป็น Shunt Regulator ที่ใช้งานได้ดีกับกระแสต่ำ) ดังรูป



รูปที่ 4.8

และหากเราต้องการให้วงจรมีประสิทธิภาพยิ่งขึ้นควรใช้ Q1 ซึ่งมี hFE สูงมากแต่จะหาทรานซิสเตอร์ซึ่งมี hFE สูง ๆ ทำได้ยาก อาจแทนที่ Q1 โดยใช้ทรานซิสเตอร์ต่อแบบดาร์ลิงตัน ดังที่ได้อธิบายมาแล้วในบทเรียนเรื่อง Series Regulator แล้วเพิ่มเติมอุปกรณ์บางตัวเพื่อความเหมาะสมในการใช้งานดังรูป

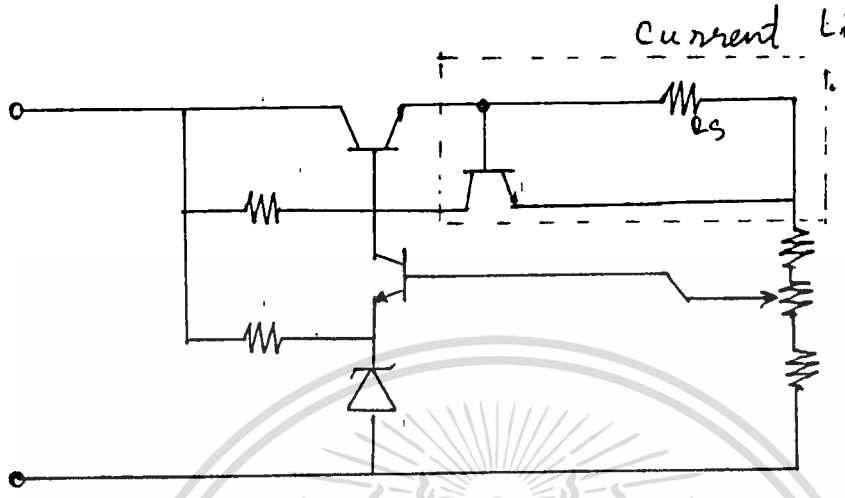


รูปที่ 4.9

Current Limitting In Power Supply

ข้อเสียประการหนึ่งของ Series Regulator คือ Pass Element จะต่ออนุกรมกับ Load ในกรณีเช่นนี้หากเกิดการชอร์ท ของโหลดจะส่งผลทำให้ Pass Element ชำรุด

เราอาจป้องกันการชำรุดของ Pass Element เนื่องจากการชอร์ทโหลดนี้ได้ โดยใส่ภาคจำกัดกระแส ซึ่งเป็นวงจรง่าย ๆ ดังรูป



รูปที่ 4.10

ค่า Rs ที่ใส่เข้าไปสามารถคำนวนได้จากสูตร

$$RS = \frac{VBE}{I \ (MAX)}$$

เช่นต้องการจำกัดกระแสสูงสุดที่ 1 Amp และ VBE = 0.7 V

$$Rs = \frac{0.7}{1}$$

$$= 0.7 \quad \text{Ohm.}$$

Monolithic Regulator

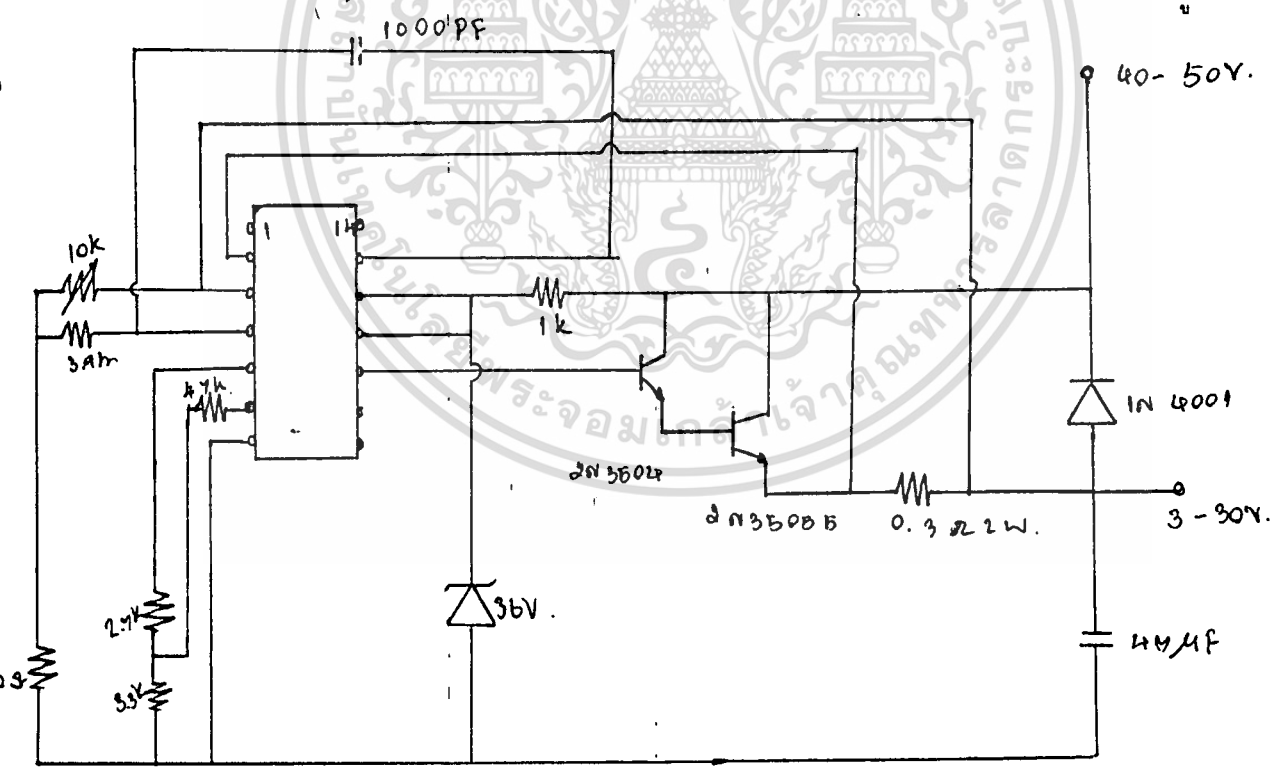
เนื่องจากการออกแบบวงจรรักษาระะดับแรงดันด้วย Discreat Element ยุ่งยาก และสิ้นเปลืองในปัจจุบันจึงมีผู้ออกแบบ Monolithic Regulator Ic สำหรับใช้งานเป็นภาครักษาระดับแรงไฟ ซึ่ง Ic ชนิดดังกล่าวสามารถแบ่งออกได้เป็น 2 กลุ่มใหญ่ ๆ

1. รักษาระดับแรงไฟค่าคงที่
2. รักษาระดับแรงไฟไปกับค่าแรงดันได้

สองชนิดนี้ยังสามารถแยกย่อยออกได้เป็น 2 ประเภท คือ รักษาระดับแรงดันบวกและรักษาระดับแรงดันลบ

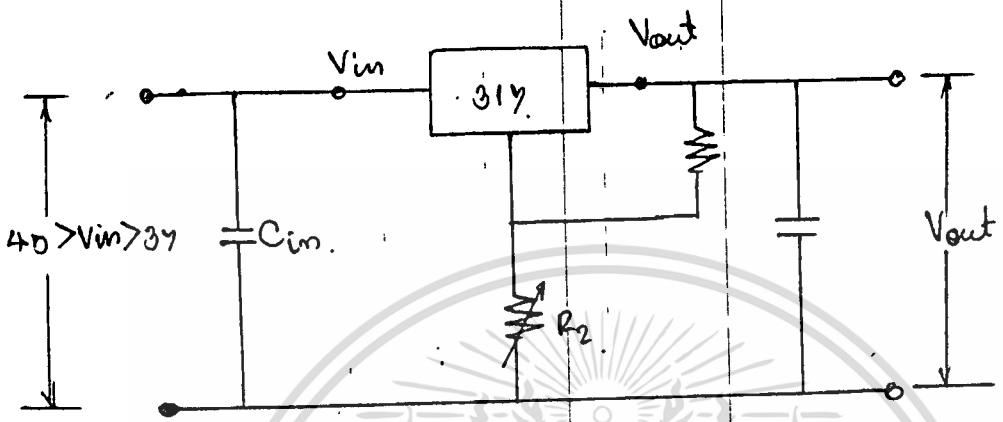
Adjustable Out Put Ic Regulator

ไอซีตระกูลนี้เป็นที่นิยมใช้ในปัจจุบัน มีด้วยกัน 2 เบอร์ คือ 723 และ 317 สำหรับเบอร์ 723 มีข้อเสียคือต้องอาศัยทรานซิสเตอร์มาต่อเป็น Pass element ดังรูป



รูปที่ 4.11

สำหรับเบอร์ 317 ในปัจจุบันเป็นที่นิยมใช้กันมากที่สุดเนื่องจากมีความสะดวกในการใช้งานและเป็นวงจรที่มีประสิทธิภาพสูงขอให้พิจารณาวงจรตามรูปซึ่งเป็นวงจรใช้งานมาตราฐาน



รูปที่ 4.12

จากรูปวงจรมีส่วนที่จะต้องคำนวนออกแบบเพียงค่าเดียวคือ R2 ซึ่งสามารถคำนวนได้โดยใช้สมาการ

$$VOUT = 1.25\ (1 + \frac{R2}{R1}) + IAdj\ R2$$

ตามปกติ IADI จะมีค่าน้อยกว่า 100 A ดังนี้ IADI R2 ซึ่งมีค่าน้อยมากและสามารถตัดทิ้งได้ในทางปฏิบัติ หรืออาจได้ว่า

$$VOUT - 1.25\ (1+ \frac{R2}{R1})$$

จากรูปวงจรเห็นว่า R2 จะเป็นตัวกำหนด VOUT ซึ่งจะมีค่าสูงสุดได้ไม่เกิน 37 V ตามคู่มือ IC และ VOUT จะมีค่าต่ำสุด ได้เท่ากับ 1.25 V เมื่อ R2 = 0 Ohm.

กรณีที่ออกแบบในวงจรดังกล่าวในสามารถปรับแรงดันได้ 1.25- 2.5 Volts เราต้องใช้ R2 เป็น Pot ที่มีค่าเท่ากับ

$$R2 = \left(\frac{25}{1.25} - 1\right) 240 = 19 \times 240 = 4.8 \text{ K Ohm.}$$

ในกรณีที่ต้องการคำนวน R2 อย่างแท้จริงก็สามารถทำได้โดยแก้สมาการ

$$VOUT = 1.25 \left(1 + \frac{R2}{R1}\right) + IODI \; R2$$

$$= 1.25 \left(1 + \frac{R2}{240}\right) + 100 \times 10^{-6} \; R2$$

เมื่อ R1 = 240 Ohm. และ Iadi max = 100 A

$$VOYT = 1.25 (1 + 4.17 \times 10^{-3} \; R1) + 100 \times 10^{-6} \; R2$$

$$= 1.25 + 5.21 \times 10^{-3} \; R2 + 100 \times 10^{-6} \; R^2$$

$$= 1.25 + 5.21 \times 10^{-3} \; R2$$

$$R2 = \frac{VOUT - 1.25}{5.21 \times 10^{-3}}$$

ในกรณีที่ต้องคำนวนค่า R2 เราไม่สามารถหา F2 ตามที่เราคำนวน ได้ดังนี้อาจทำได้อีกกรณีหนึ่งคือ กำหนดค่า R2 และ VOUT แล้วคำนวน หา R1 ก็ได้เช่น ต้องการ VOUT MAX = 30 V. โดยใช้ POT 5 K Ohm.

$$30 = 1.25 \left(1 + 5 \times 10^{3}\right) + 100 \times 10^{-6} \times 5 \times 10^{3}$$

$$\frac{}{R1}$$

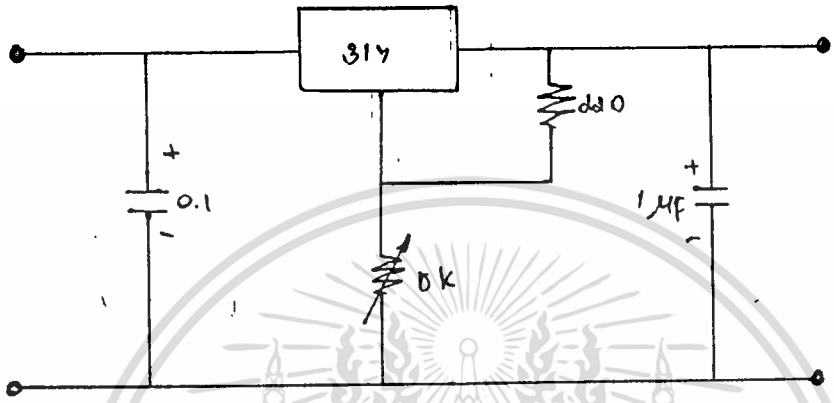
$$= 1.25 + \frac{6250}{R1} - 0.5$$

$$\frac{6250}{R1} = 30 - 1.25 - 0.5$$

$$= 28.25$$

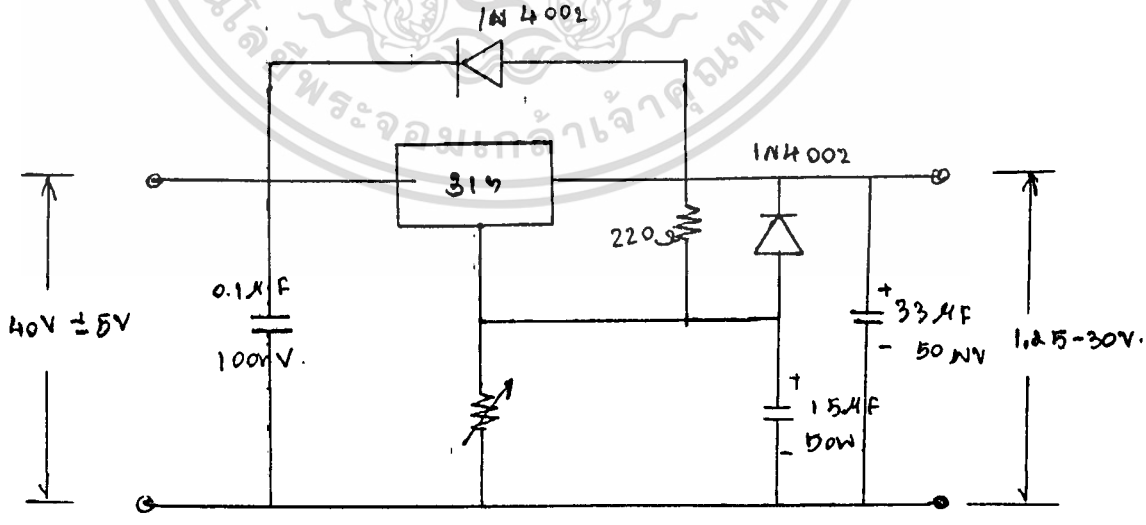
$$R1 = \frac{6250}{28.25} = 221.23 \text{ Ohm.}$$

เลือกใช้ R1 = 221 Ohm. +- 5%



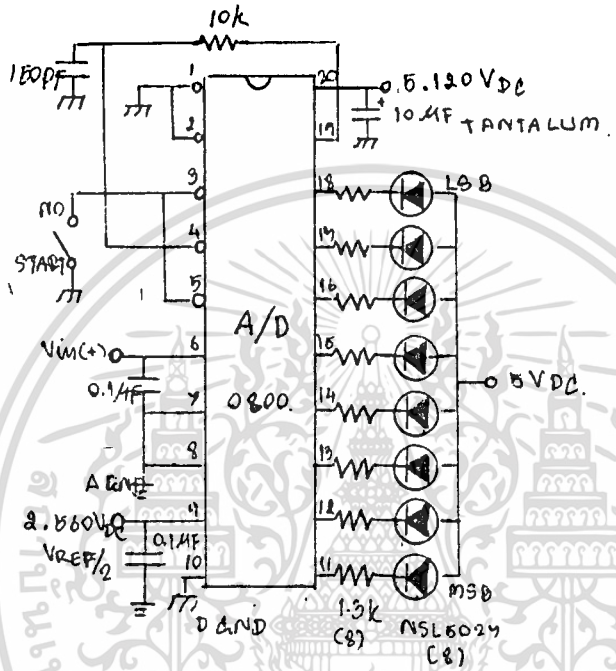
รูปที่ 4.13

สำหรับวงจรในทางปฏิบัติอาจมีอุปกรณ์มากกว่าวงจรมาตรฐานเพื่อป้องกันสิ่งอื่น ๆ เช่น Transistant Bock E.M.F ได้สมบูรณ์ จากรูปเป็นตัวอย่างวงจร



รูปที่ 4.14

วงจร Analog to digtal

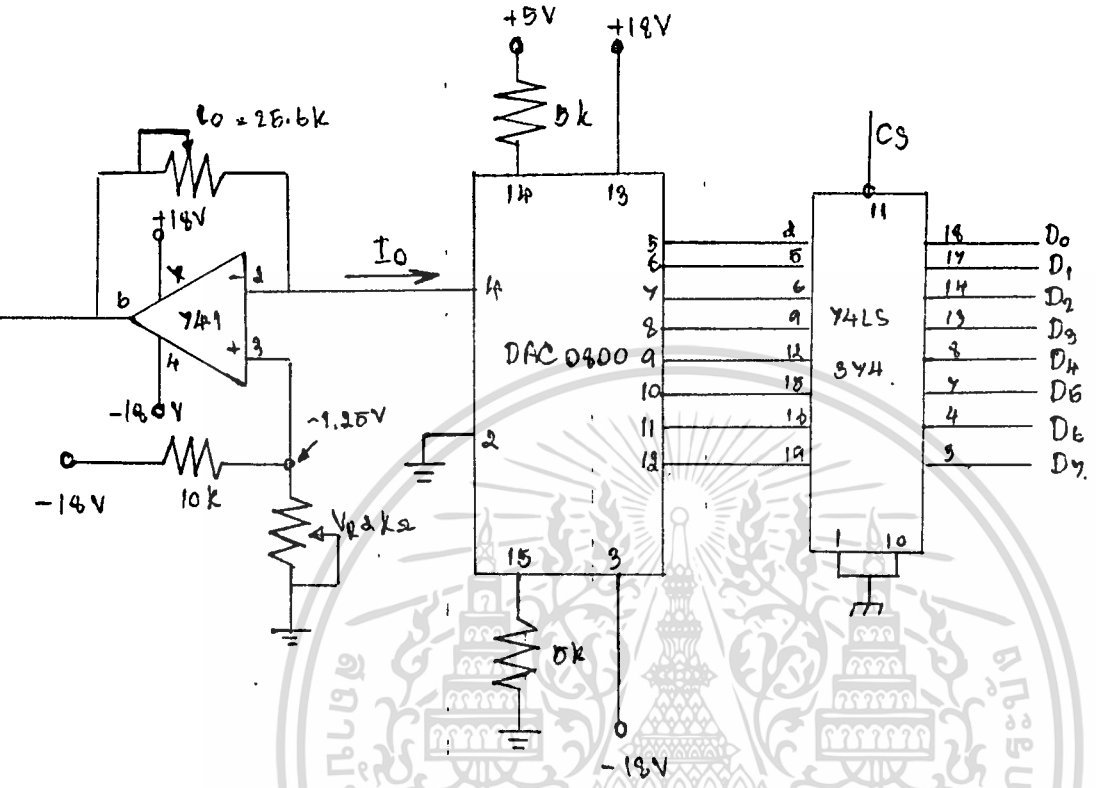


รูปที่ 4.15

จากรูป เป็นวงจร Analog to digtal 0803 ทำหน้าที่แปลงสัญญาน Analog เป็นสัญญาณ digital ขนาด 8 บิท ค่าแรงดัน supply สูงสุด 6.5 โวลท์ Conversion time100us ความถี่ clock 640kH สามารถป้อน รียีย voltage ได้ 0.5 โวลท์

เมื่อเราต้องการให้ ADC 0803 ทำหน้าที่เปลี่ยนค่า Analog เป็น digital โดยทำให้ขา cs active low จากนั้นสัญญาณจะถูกส่งไปยังไมโครโปรเซสเซอร์

วงจร Digital to analog



รูปที่ 4.16 วงจรดิจิตอลเปลี่ยนเป็นอนาล็อก

จากวงจร digital to analog ใช้ไอซี เบอร์ 0800 การทำงานโดยเริ่มจากนำค่า digital ที่ต้องการมาเก็บไว้ใน ไอซี เบอร์ 74LS 374 จะทำงานช่วง active low จากนั้นค่า digital จะถูกส่งมายัง ไอซี 0800 เพื่อทำการแปลงสัญญาณ digital เป็นสัญญาณ analog ซึ่งทางเอาท์พุทของไอซี 0800 นั้นจะมีวงจร current to voltag เพื่อทำการเปลี่ยนแปลงกระแสเป็นแรงดัน ไปป้อนแก่ขา 1 ของไอซี LM 317

เนื่องด้วย ไอซี LM 317 มีค่า Voltage reference เมื่อแรงดันที่ขา 1 เท่ากับ 0 Volts จะมีแรงดันเอาท์พุท 1.25 Volts ดังนั้นเพื่อให้แรงดันเอาท์พุทเท่ากับ 0 Volt เราจึงให้แรงดันที่ขา 1 ของ LM 317 เป็นลบ เราจึงออกแบบวงจร current tovoltage อยู่ในรูป defferencetialamp โดยมีอินพุท 2 อินพุท ต่อเข้าที่ OP AMP LM 741 ที่ขา 2 และขา 3 จากการทดลองเราปรับให้ เอาท์พุทของ OP-AMP LM 741

ข้อกำหนดให้ $V_{ref} = \frac{V_{cc}}{2} = \frac{5}{2} = 2.5$ V.

และ 1 STEP $= \frac{5}{256} = 19.5$ mv. 20 mv

DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	VIN (VIN)
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0.02
0	0	0	0	0	0	1	0	0.04
0	0	0	0	0	0	1	1	0.06
0	0	0	0	0	1	0	0	0.08
0	0	0	0	0	1	0	1	0.10
0	0	0	0	0	1	1	0	0.12
0	0	0	0	0	1	1	1	0.14
1	1	1	1	1	1	1	1	5

จากรูปเป็นวงจรทดลอง analog to digital โดยป้อนอินพุทที่รู้ค่า ซึ่งค่าของสัญญาณเอาท์พุทที่ได้จะแสดงผลด้วย LED ลองเปรียบค่าแรงดันอินพุท และสังเกตุการเปลี่ยนแปลงของ LED และเมื่อป้อนอินพุท 5 VOLTAGE ทำให้ค่าเอาท์พุทเป็น 11111111

อยู่ระหว่าง -1.2 ถึง 16.7 Voltage เพื่อที่จะให้สามารถปรับแรงดันเอาท์พุทของภาค power regulator อยู่ในช่วง 0- 18 Voltage

จากการทดลอง

กำหนดให้ $V_{ref} = \frac{V_{cc}}{2} = \frac{5}{2}$

$1\ STEP = \frac{5}{256} = 19.5\ mv - 20\ mv$

$I_{REF} = \frac{V_{REF}}{R_{REF}} = \frac{5V}{5K} = 1\ ma$

$1\ STEP = IO = I_{REF}. \frac{XXX}{256}$

$= 1\ MA. \frac{1}{256} = 3.90625$

เมื่อ IO = 0

คือ IO = 1 MA. $\frac{0}{256}$ = 0 นั่นเอง

และค่า VO = VRL - VREF

= - 1.25 V

เมื่อ IO = 3.90625 ที่ STEP 1

IO = 1 mA. $\frac{1}{256}$ = 3.90625

และเราต้องการ VO = -1.15 V.

เพื่อจะทำให้ Regulator 317 มีค่า Vout คงที่ 0.1 V. คือ 1 STEP ให้ค่าเปลี่ยนแปลงที่ 0.1 นั่นเอง

VO = -1.15 V.

VREF = 1.25

VRO = VO + VREF = 0.1 V.

IO = 3.90625 = IRL

$RL = \frac{VRK}{IO} = \frac{0.1\ V}{3.90625} = 25.6\ K\ OHM$

เมื่อคำนวณหาค่า RO ได้แล้วคือ 25.6 K

ที่ STEP 2

$$IO = Iref \cdot \frac{2}{256}$$

$$= 7.8125 \text{ ma}$$

$$Vro = IO \; RO$$

$$= 0.2 \text{ V.}$$

$$VO = Vro - Vref = 0.2 - 1.25$$

$$= -1.05$$

Vout ของ Regalator 317 จะเท่ากับ

$$vout = -1.05 + 1.25 \text{ V}$$

$$= 0.2 \text{ V}$$

ที่ STEP 3

$$IO = Iref \cdot \frac{3}{256}$$

$$= 1 \cdot \frac{3}{256}$$

$$= 11.71875$$

$$Vro = I_{ref} \cdot RO$$

$$= 0.3 \text{ V.}$$

$$VO = Vro - Vref$$

$$= 0.3 - 1.5 \text{ V}$$

$$= -0.95 \text{ V}$$

Vout ของ Regulator 317 เท่ากับ

$$Vout = -0.95 + 1.25 \text{ V}$$

$$= 0.3 \text{ V นั่นเอง}$$

ต้องการ Vout ออก 18 v

STEp 1 ได้คือ 180 step

IO = Iref $\frac{180}{256}$

= 703.125

Vo = Vro - Vref

= 18 - 1.25 v

= 16.75 V

Vout ของ Regulator 317 จะเท่ากับ

Vout = 16.75 + 1.25 V.

= 18 V

ผลการทดลอง

r1 = 25.6 Vref = 5 Rref = 5 Iref = 1

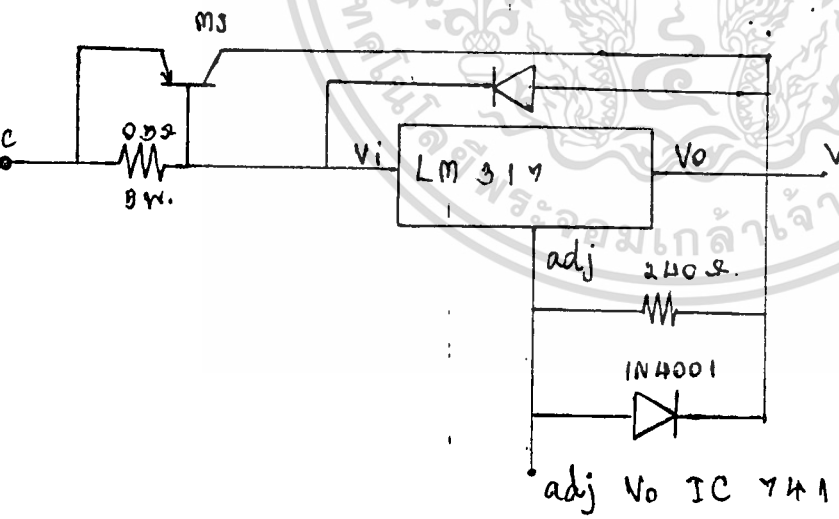
STEP	VO	VO	VOUT	VOUT
0	-1.25	-1.25	0	0
1	-1.14	-1.15	0.12	0.1
2	-1.06	-1.05	0.21	0.2
3	-0.93	-0.95	0.34	0.3
4	-0.82	-0.85	0.42	0.4
5	-0.73	-0.75	0.53	0.5
6	-0.62	-0.65	0,64	0.6
7	-0.53	-0.55	0.72	0.7
8	-0.42	-0.45	0.80	0.8
180	16.70	16.75	17.9	18

การทำงานของ Programmable power supply

การทำงานของระบบควบคุมแหล่งจ่ายไฟด้วยไมโครโปรเซสเซอร์เป็นวงจร Regulato ที่สามารถปรับแรงดันเอาท์พุทได้ระหว่าง 0 -18 Volt โดยการทำงานของวงจาจรทั้งหมด จะถูกควบคุมด้วยไมโครโปรเซสเซอร์ ขั้นตอนการทำงานโดยเริ่ม จากภาค Power Regulator ทำหน้าที่เปลี่ยนไฟ 220 V เป็น 0- 18 V.D.C โดยค่าของแรงดันเอาท์พุทจะถูกควบคุมด้วยไมโครโปรเซสเซอร์เบอร์ Z-80 ซึ่งค่าของแรงดันที่เราต้องการ จะถูกป้อนเข้าไปทาง KEY Board โดยมี LED และแสดงค่าแรงดันที่เราต้องการ และค่าแรงดันนี้จะถูกส่งไปยังวงจร Digital to analog

วงจร Digital to anatog (DAC) จะทำหน้าที่เปลี่ยนสัญญาณ Digital เป็นสัญาณ analog เพื่อเป็นสัญญาณไปปรับ Power regulator เพื่อให้ค่าเอาท์พุทตามต้องการ ค่าแรงดันเอาท์พุทจะถูกส่งไปยังวงจร analog to digtal เพื่อส่งสัญญาณ digital มายังไมโครโปรเซสเซอร์ ตัวไมโครโปรเซสเซอร์ จะทำการเปรียบเทียบค่าว่าแรงดันเอาท์พุทถูกต้องหรือไม่ ถ้าไม่ถูกต้องก็จะทำการปรับให้ถูกต้องตามที่ต้องการ การทำงานแต่ละวงจรเป็นดังนี้

A. POWER REGULATOR



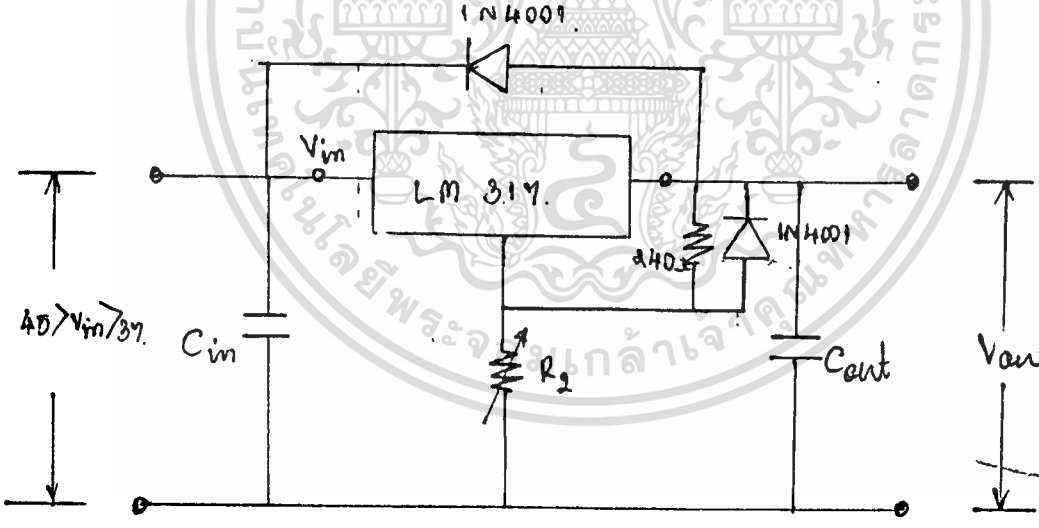
รูปที่ 4.17 วงจรเร็กกูเลเตอร์ที่ใช้ในวงจร

จากรูป A เป็นวงจร power ergulator โดยใช้เบอร์ LM 317 เป็น IC ขนาด 1.5 A ทำงานที่อุณหภูมิตั้งแต่ 0 -125 c กระแส adjust 50 ไมโครแอมป์ ความแตกต่างระหว่างแรงดันอินพุทกับแรงดันเอาท์พุทไม่น้อยกว่า 3 V. ไม่เกิน 40 V

แรงดันทางอินพุทของ LM 317 จากวงจร rectifier และมี Capacitor 0.1 ทำหน้าที่ by pass ความถี่สูงมี Resistor 10 K ทำหน้าที่ Discharge กระแสขณะปิดเครื่อง ค่าแรงดันเอาท์พุทอยู่ในช่วง 0 - 18 V. โดยมี Capacitor 220 mF เป็นตัว Filter ค่าแรงดันเอาท์พุท จะถูกส่งไปยังวงจร analog to digital ผ่านวงจร Voltage divider

เนื่องด้วย LM 317 มี Reference Voltag (ระหว่างขา 1 -3) อยู่ 1.25 Volts เพื่อต้องการให้แรงดันเอาท์พุท เท่ากับ 0 -18 V. จึงต้องป้อนแรงไฟเข้าที่ขา 1 อยู่ในช่วง - 1.25 - 16.75 V.

วงจรที่ใช้ในการทดลอง



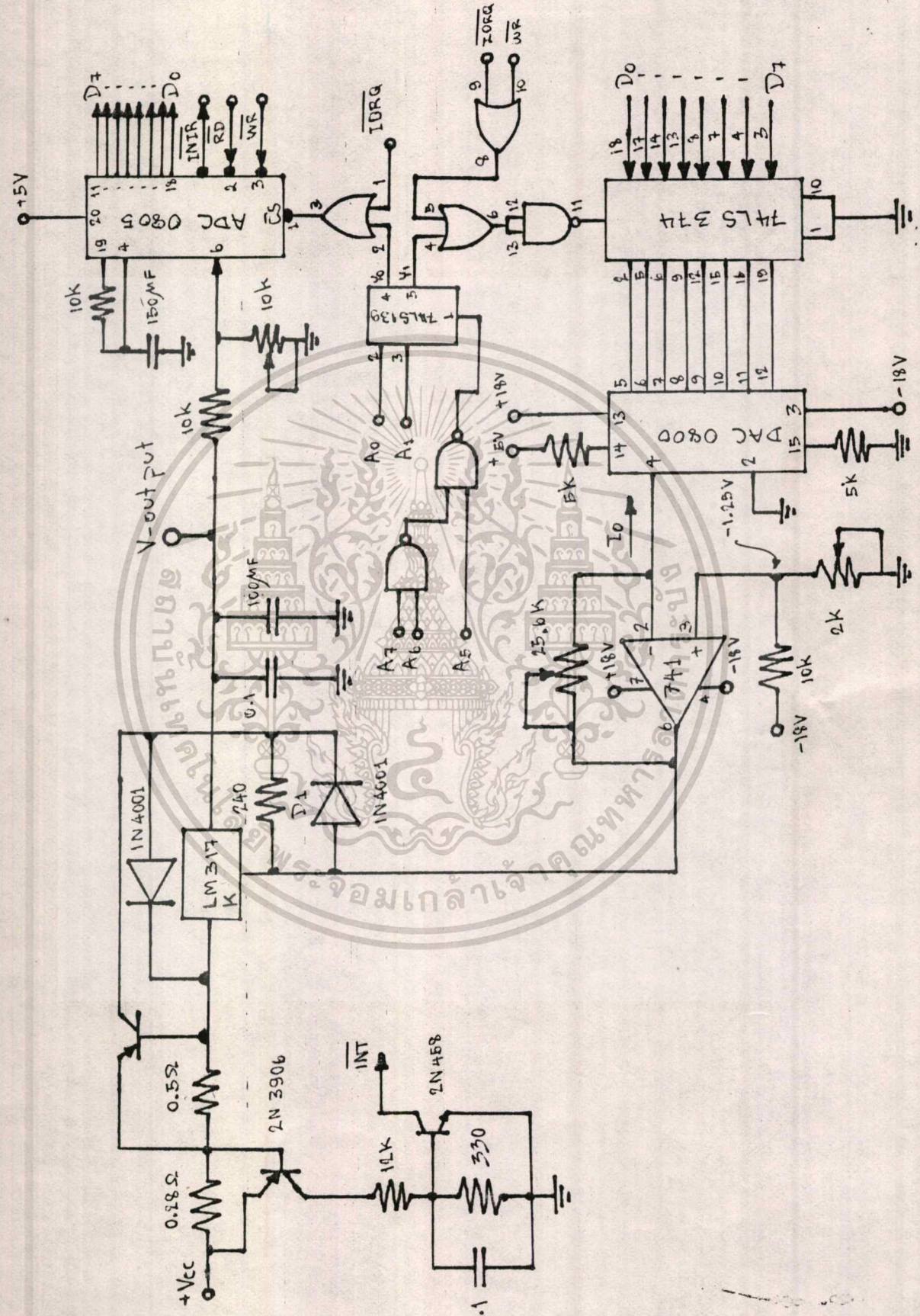
รูปที่ 4.18

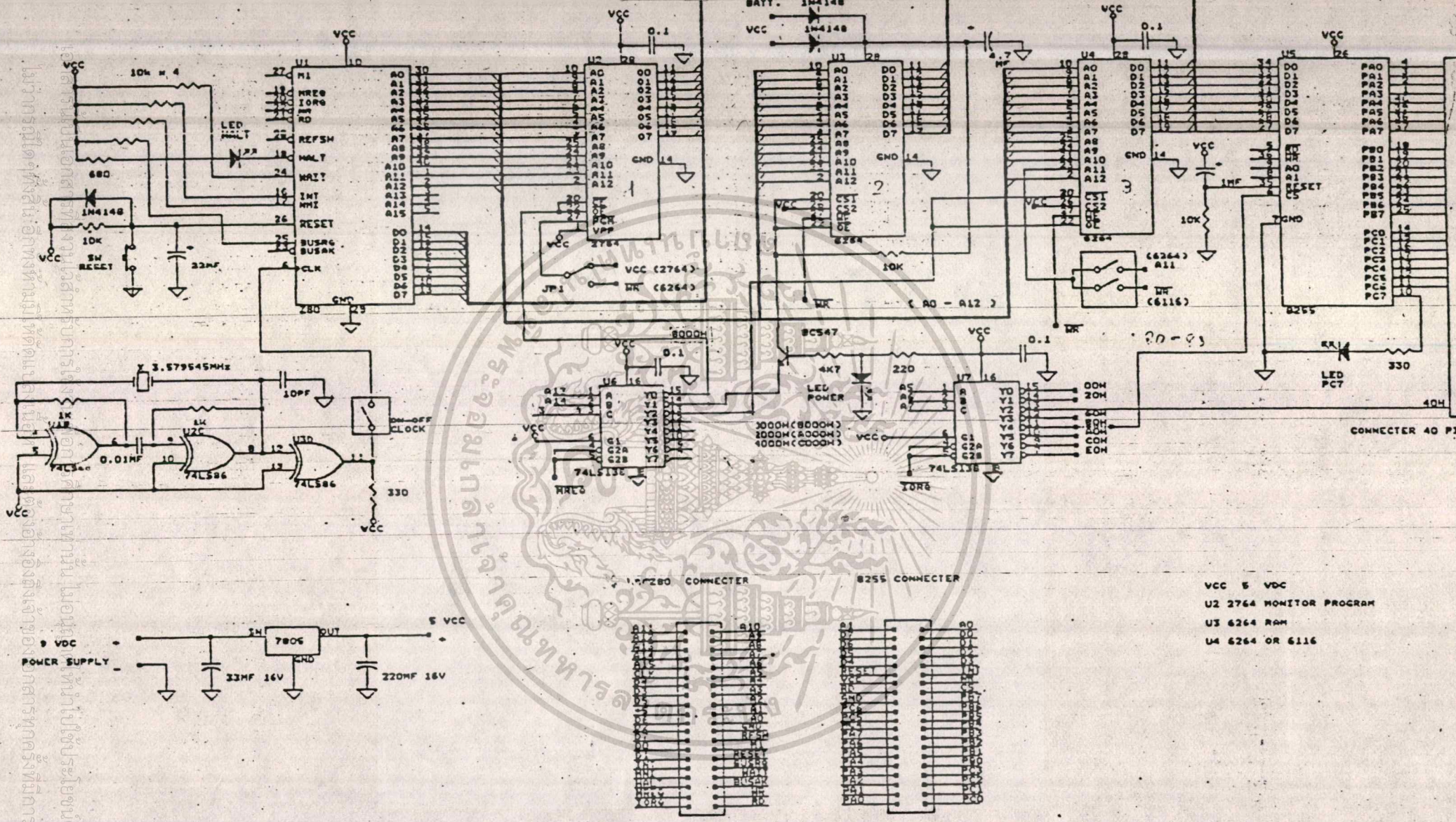
จากรูป B เป็นวงจรที่ใช้ในการทดลอง ทำการทดลองป้อน ปรับ ดูการเปลี่ยนแปลงของแรงดัน เอาท์พุท ซึ่งสามารถทำงานได้ดังนี้

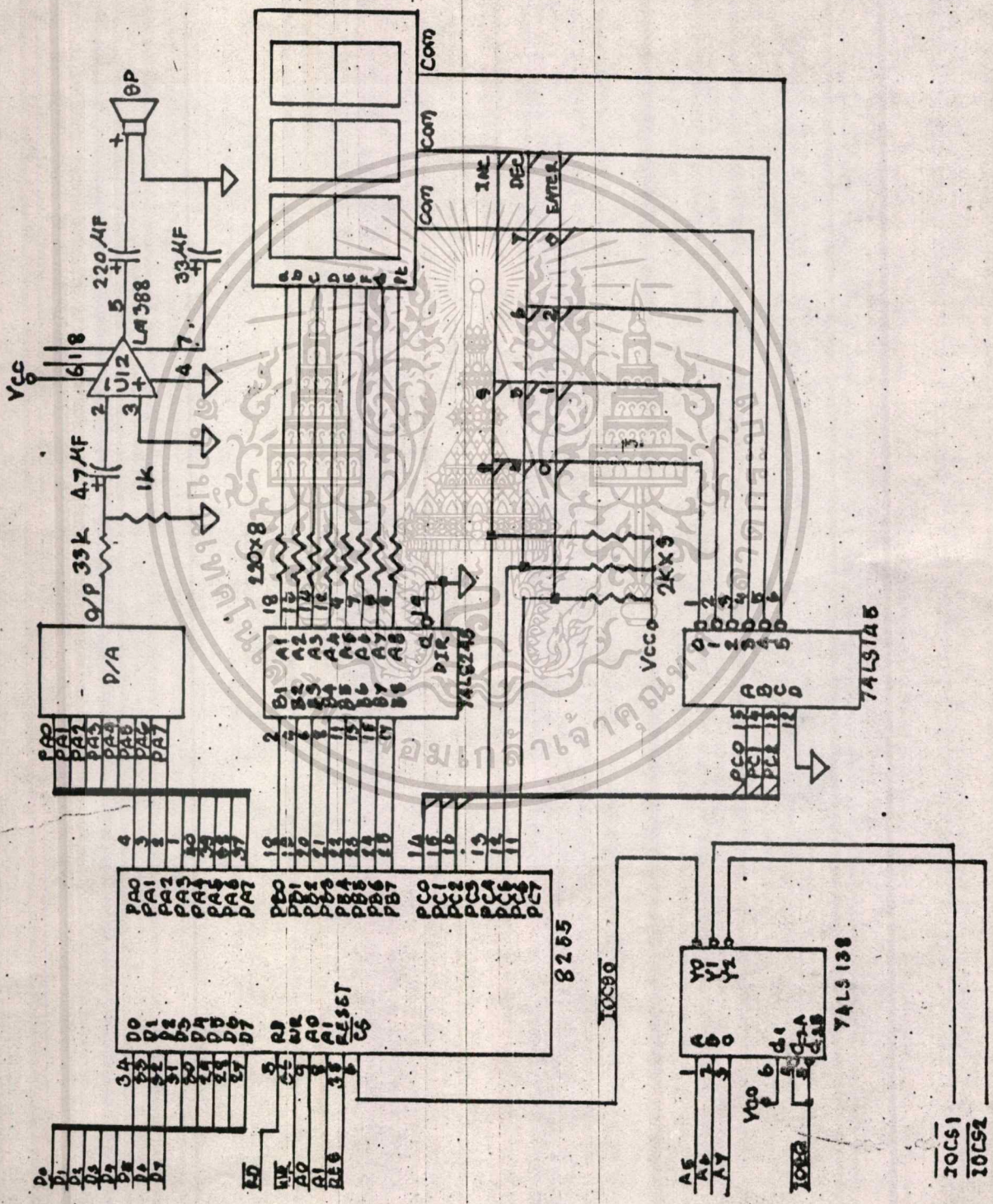
$$VC = 1.25 \left(1 + \frac{R2}{R1} \right) + IADJ \; R2$$

จากวงจรไดโอด D1 นั้นเป็นตัวป้องกันจากการที่ Cc ทำการ discharge ในกรณีที่อินพุทลัดวงจร (short circuit) ส่วนไดโอด D2 นั้นป้องกันเมื่อเอาท์พุทลัดวงจร

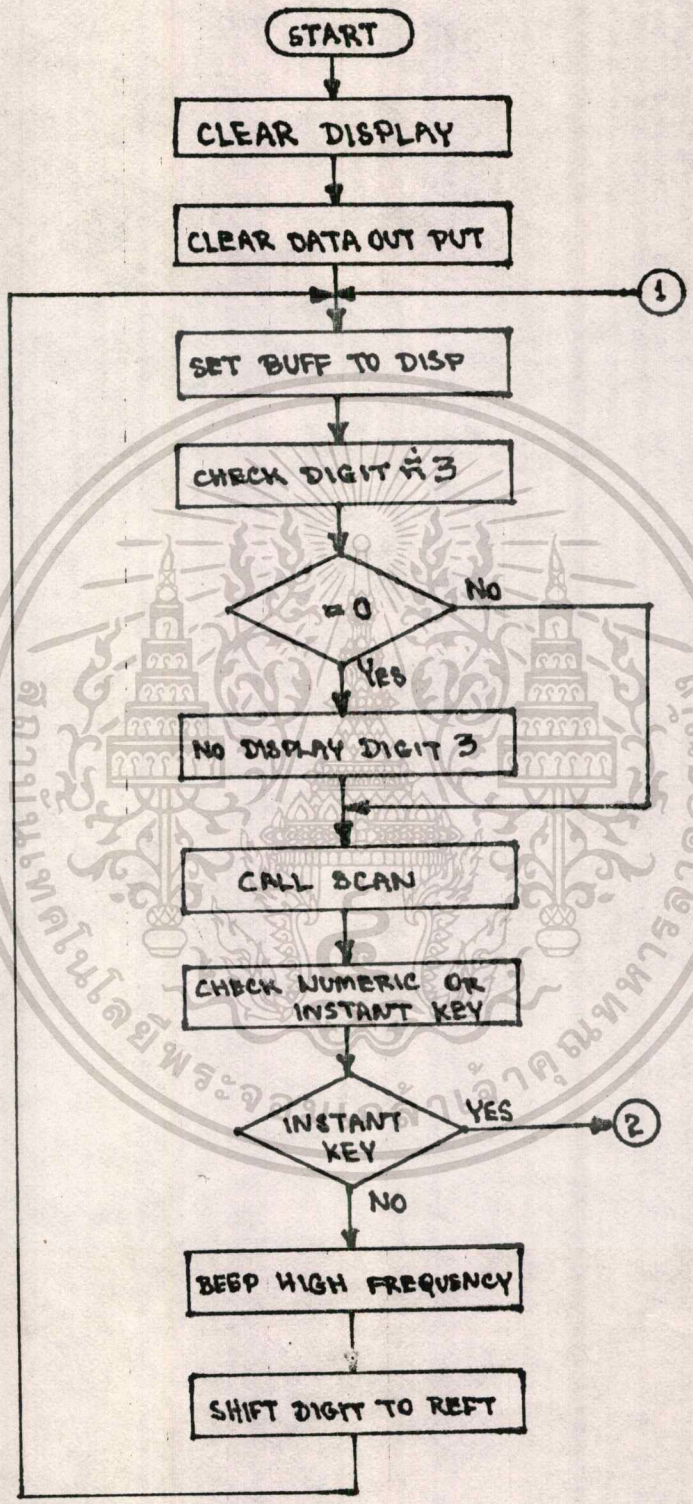
วงจร CONTROL POWER SUPPLY

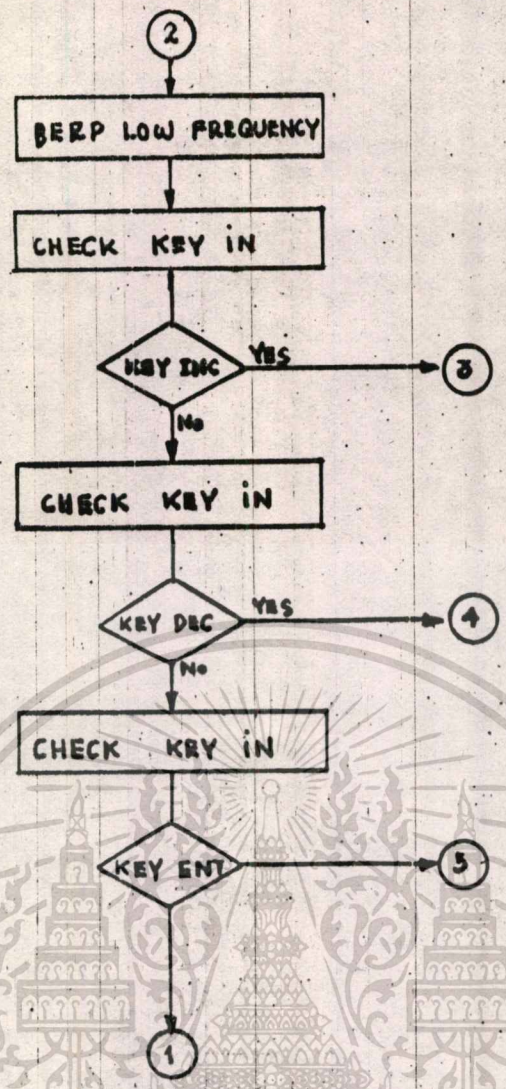



VCC
10K x 4
LED HALT
680
1N4148
10K
SW RESET
33MF
Z80
M1
MREQ
IORQ
RD
REFSH
HALT
WAIT
INT
NMI
RESET
BUSRQ
BUSAK
CLK
GND
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15
D0 D1 D2 D3 D4 D5 D6 D7
U1
U2
2764
U3
6264
U4
6264
U5
8255
0.1
4.7 MF
BATT.
1N4148
JP1
VCC (2764)
WR (6264)
WR
(A0 - A12)
(6264) A11
WR (6116)
1MF
10K
RESET
GND
PA0 PA1 PA2 PA3 PA4 PA5 PA6 PA7
PB0 PB1 PB2 PB3 PB4 PB5 PB6 PB7
PC0 PC1 PC2 PC3 PC4 PC5 PC6 PC7
LED PC7
330
404
CONNECTER 40 PIN
8000H
BC547
4K7
220
LED POWER
U6
U7
74LS138
MREQ
IORQ
8000H(8000H)
2000H(A000H)
4000H(C000H)
00H
20H
40H
60H
80H
A0H
C0H
E0H
Y 3.579545MHz
10PF
ON-OFF CLOCK
1K
74LS86
74LS86
74LS86
0.01MF
330
9 VDC
POWER SUPPLY
7805
IN
OUT
GND
33MF 16V
220MF 16V
5 VCC
Z80 CONNECTER
8255 CONNECTER
VCC = 5 VDC
U2 2764 MONITOR PROGRAM
U3 6264 RAM
U4 6264 OR 6116

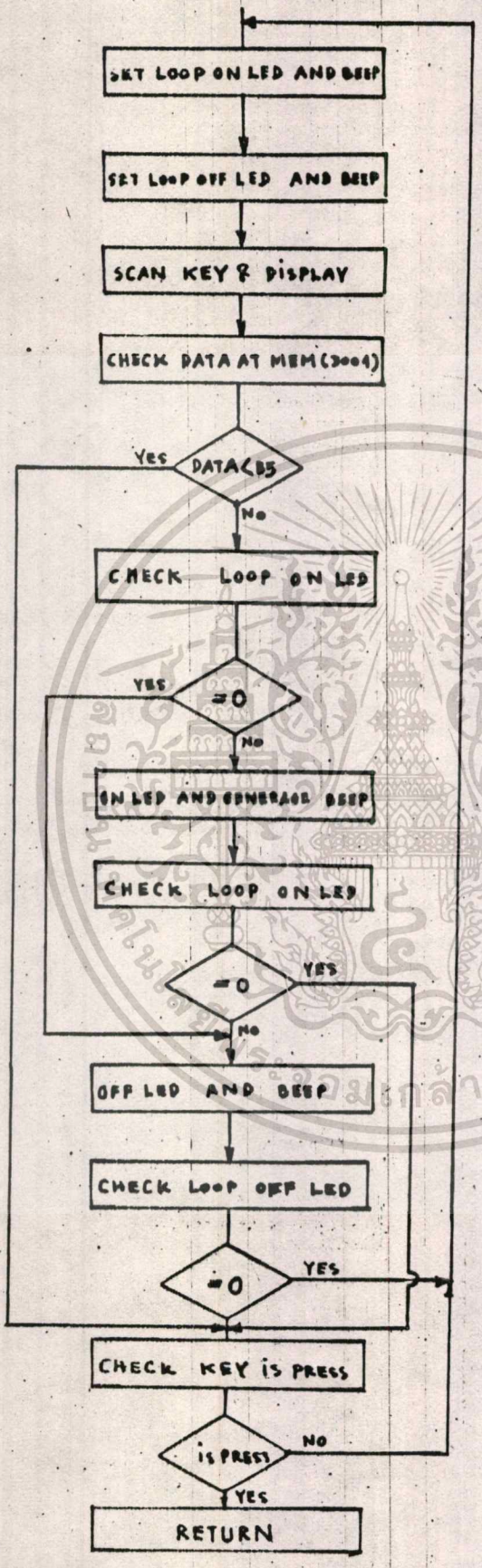


FLOW CHART CONTROL POWER SUPPLY

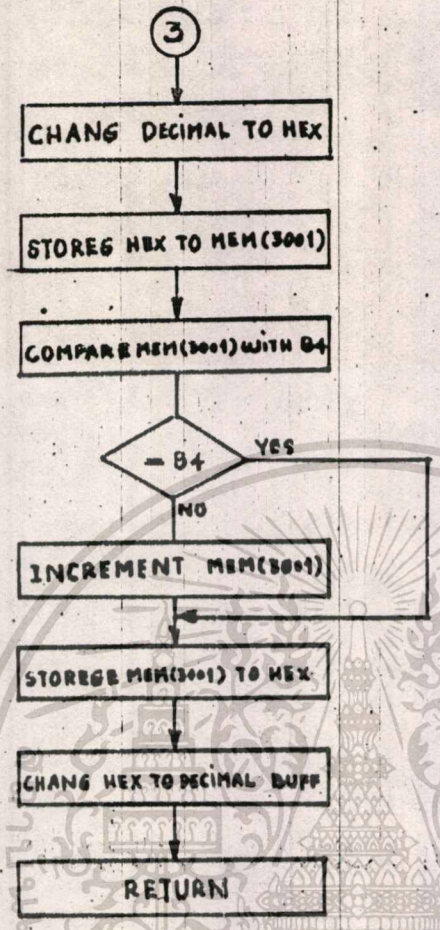




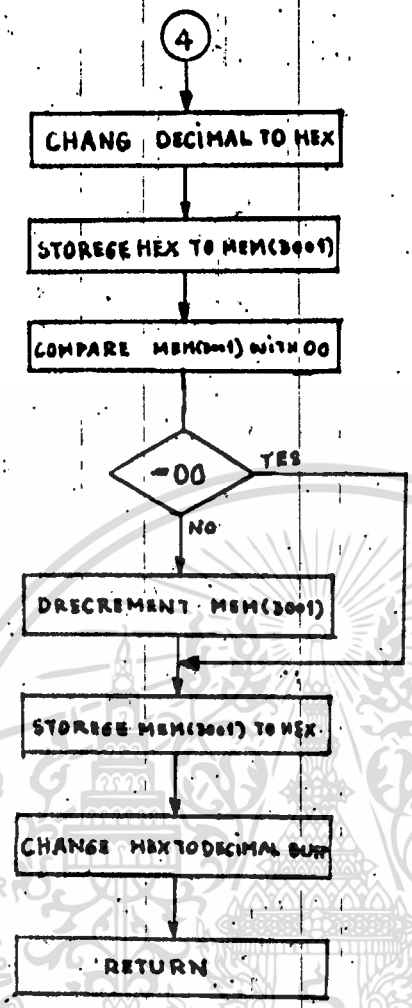
SCAN KEY & DISPLAY



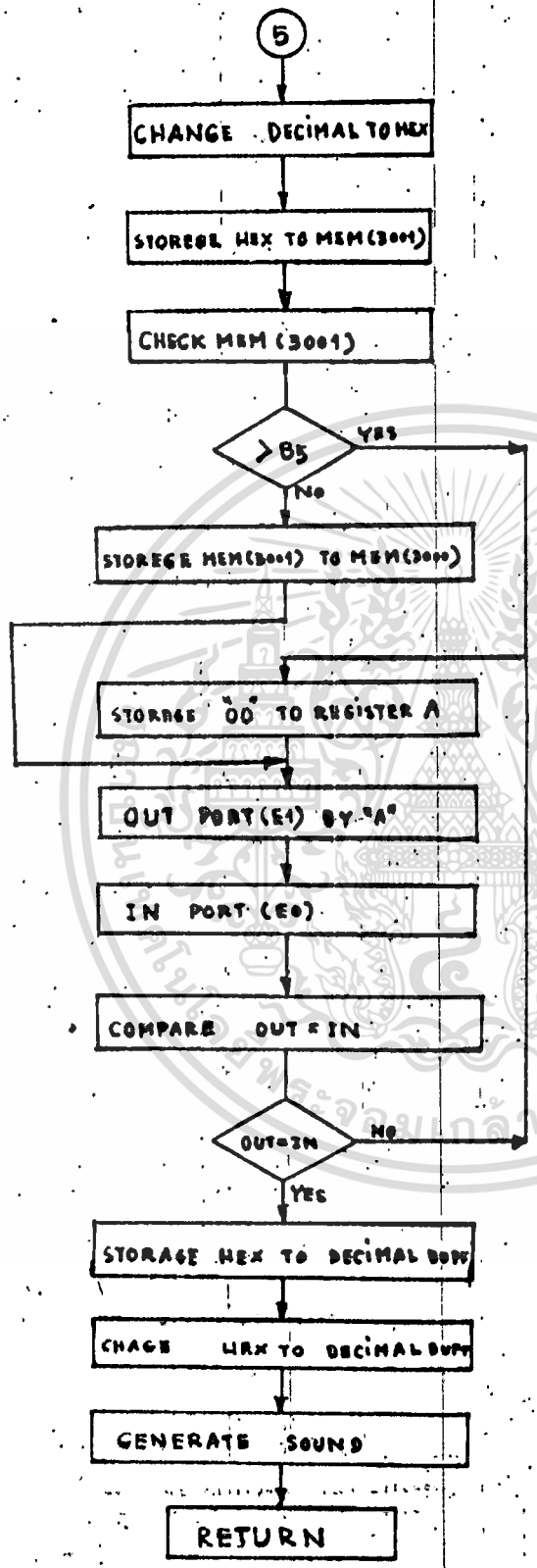
INCREMENT FUNCTION



* DRECREMENT FUNCTION *



* ENTER FUNCTION *



PROGRAM CONTROL POWER SUPPLY

```
ORG   4000
        DISPLAY   BUFFER      =    3FE7H-3FEEH
        BUFFER                =    3FEFH-3FF1H
        KEY       CODE        =    3FFBH
        SYSTEM    FLAG        =    3FFDH
        DECIMAL   BUFFER      =    3FF4H-3FF6H
        HEX       BUFFER      =    3FF2H-3FF3H

4000 START:   LD  A,03H       3E03
        RST   10H             D7          ;CLEAR  DISPLAY
        LD  HL,3FFDH          21FD3F
        RES 4,(HL)            CBAC        ; NO  CHECK  INSTANT KEY
        XOR A                 AF
        OUT A,E1              D3E1
        LD (3000H),A          320130
        LD (3001H),A          320130
4011   KK     LD A,(3FF4H)          3AF43F
        LD (3FF1H),A          32F13F
        LD A,(3FF5H)          3AF53F
        LD (3FEFH),A          32EF3F    : LOAD DECIMAL BUFF TO BUFF
```

```
        LD A,07H          3E07
        RST  10H          D7        :CHANGE BUFF TO DISP BUFF
4020        LD A,(3FF5H)      3AF53F
        AND 0FH           E60F
        JR NZ,KK0         2004      : CHECK COLUMN =3
        XOR  A            AF
        LD (3FEAH),A      32EA3F
  KK0     LD HL,3FEBH       21EB3F
        SET 7,(HL)        CBFE      : "." TO SECOND DIS
        CALL  SCAN        CD004A
        BIT 4,A           CB67
        JR NZ,KK3         2011      : CHECK NUMERIC OR INSTANT
        LD A,0EH          3E0E
        RST 10H           D7        : BEEP HIGH FREQUENCY
        LD A,(3FF4H)      3AFB3F
        LD HL,3FF4H       21F43F
        RLD               ED6F
        LD (3FF5H),A      32F53F
  KK2     JP KK             C31140

404B  KK3     LD A,1EH          3E1E     :BEEP LOW FREQUENCY
        RST 10H           D7
        LD A,(3FFBH)      3AFB3F
        CP 14H            FE14     : CHECK KEY "INC"
        CALL Z,INC        CC584A
        LD A,(3FFBH)      3AFB3F
        CP 12H            FE12     : CHECK KEY "DEC"
```

```
        CALL Z,DEC      CC774A
        LD A,(3FFBH)    3AFB3F
        CP 10H          FE10    :CHECK KEY "ENTER"
        JR  KK2         18E0

4A00 SCAN *** LD A,FF           3EFF    :SET DATA LOOP ON
        LD (3000),A     320330
        LD A,FF         3EFF    :SET DATA LOOP OFF
        LD (3004),A     320430

4A0A ****   LD A,04             3E04
        LD B,04         0604
        RST 10          D7      : DELAY=B
        LD A,FF         3EFF
        LD (3001),A     32FB3F  :CLEAR KEY CODE
        LD A,05         3E05
        RST 10          D7      :SCAN KEY & DISPLAY
        LD A,(3001)     3A0130  : CHECK DATA OUT

        SUB B5          D6B5
        JPC  **         DA524A  IF <
        LD A,(3003H)    3A0330
        CP 00           FE00
        JP Z ***        CA3E4A
        LD A,1E         3E1E
        RST             D7
        LD A,01         3E01
```

```
        OUT 01,A        D301
        LD A,06         3E06
        OUT 02,A        D302
        LD HL ,3003H    210330
        DEC (HL)        FE00

4A38        JPNZ **         CA524A   IF LOOP LAMP ON GO TO ***
4A3B        LD A,00         3E00
        OUT 01,A        D301
        LD A,06         3E0C
        OUT 02,A        D302
        LD HL,3004      210430
        LD A,(3004)     3A0430
        CP 00           FE00
        JP Z ***        CA004A   IF LOOP LAMP OFF GO TO ***
4A52  **    LD A,(3FFBH)    3AFB3F
        CP FF           FEFF
        JP Z ****       CA004A
        RET             C9

4A5B INC    LD A,0B         3E0B
        RST 10          07
        LD HL,3FF2      21F23F
        LD A,(HL)       7E
        LD (3001H),A    320130
        CP B4           FEB4
```

```
        JR Z *          2804
        LD HL,3001H     210130
        INC (HL)        34
    *     LD A,(3001)     3A0130
        LD (3FF2),A     32F23F
        LD A,0A         3E0A
        RST  10         D7
        RET             C9

4A77 DEC     LD A,0B          3E0B
        RST 10          D7
        LD HL,3FF2H     21F23F
        LD A,(HL)       7E
        LD ,(3001),A    320130
        CP  00          FE00
        JP Z  ##        CA8A4A
        LD HL,3001H     210130
        DEC (HL)        35
     ##  LD A,(3001)     3A0130
        LD (3FF2H),A    32F23F
        LD A,0A         3E0A

4B4F ENTER   LD A,0B          3E0B
        RST 10          D7
        LD HL,3FF2H     21F23F
```

	LD A,(HL)	7E
	LD (3001),A	320130
	SUB B4	D6B5
	JPC $	D2674B
	LD A,(3001H)	3A0130
	LD (3000H),A	320030
	JUMP $$	C36C4VB
$	LD A,00	3E00
	LD (3000),A	320030
$$	OUT E1,A	D3E1
	LD B,A	47
	IN A,E0	DBE0
	CP B	B8
	JR NZ $	20F1
	LD HL,3FF2	21F23F
	LD (HL),A	77
	LD A,0A	3E0A
	RST 10	D7
	LD A,15	3E15
	RST 10	D7
	RET	C9

วิธีใช้เครื่อง PROGRAMMABEE POWER SUPPLY

1. เปิดเครื่องเครื่องจะเป็น AUTOMATIC ขึ้นมาแต่ VOLTAGE ยังไม่ออกมา เครื่องจะรอจนกว่าเราจะกด KEY ENTER จึงจะให้ VOLTAGE ที่ OUT PUT เมื่อเราต้องการ VOLTAGE 5 VOLT ให้กด KEY 5, KEY 0 เครื่องจะโชว์ที่หน้าปัดเป็น 5.0 ตอนนี้ 5 VOLT ยังไม่ออกมา OUT PUT เราต้องกด KEY ENTER เครื่องถึงจะให้ OUT PUT 5 V ที่ OUT PUT ให้และถ้าเราเปลี่ยน VOLTAGE สมมติว่าเปลี่ยนไป 15 VOLT ก็ให้กด KEY 1 และกด KEY 5 แล้ว KEY 0 หน้าปัดจะโชว์ 15.0 แล้วเราก็กด KEY ENTER อีกทีหนึ่ง เครื่องก็จะให้ OUT PUT ออกมา

2. ถ้าต้องการ VOLTAGE เพิ่มขึ้นสามารถใช้ KEY INCREMENT ได้โดยกดค้างไว้ค่าของ VOLTAGE จะเพิ่มขึ้นทีละ STEP ละ 0.1 VOLT และเมื่อได้ค่าที่เราต้องการแล้วให้กด KEY ENTAGE จะทำให้ค่าที่โชว์ที่ DISPLAY จะไปออกที่ OUT PUT

3. ถ้าต้องการ VOLTAGE ลดลงสามารถใช้ KEY DECREMENT ได้โดยกดค้างไว้ค่าของ VOLTAGE จะลดลงทีละ STEP = 0.1 V เช่นกันและเมื่อได้ค่าที่ต้องการแล้วปล่อย KEY นี้เสร็จแล้วให้กด KEY ENTER จะทำให้ค่าที่โชว์ที่ DISPLAY จะออกไปที่ OUT PUT

4. ถ้าในกรณีที่เรากด VOLTAGE เกินค่าที่กำหนดไว้คือ 18VOLT จะทำให้ค่าที่โชว์ DISPLAY จะเป็น 0.0VOLT แล้วกระพริบด้วย และเกิดเสียงตามไฟกระพริบนั้น แสดงว่าค่าที่ป้อนเข้าไปนั้นเกินค่าที่กำหนดไว้แล้วจะทำให้ OUT PUT มีไฟออก 0.0 VOLT นั้นเอง

อุปสรรคในการทำงาน

สิ่งที่สำคัญที่เจอคือ การทดลองไม่เป็นไปตามทฤษฎี เช่นวงจร analog to digtal และ วงจร digtal to analog และการใช้ ไมโครคอมพิวเตอร์เข้าไปควบคุม สัญญาณต่างๆ จะต้องไม่มีความผิดพลาดซึ่งถ้าเกิดการผิดพลาดขึ้นมาจะทำให้ไมโครคอมพิวเตอร์ทำงานไม่ถูกต้อง

สรุป

ชุดควบคุมเหล่งจ่ายไฟด้วยไมโครโปรเซสเซอร์ จัดได้ว่าเป็นอุปกรณ์ทางอีเล็คทรอนิคส์ที่ประยุกต์ขึ้นมาใหม่ชนิดหนึ่ง เพราะโดยทั่วไปตามท้องตลาด REGULATOR จะเป็นแบบทั่วไป ในการปรับแรงดันเอาท์พุท จะใช้มือปรับให้ได้แรงเอาท์พุทตามต้องการแต่ชุดควบคุมเหล่งไฟด้วยไมโครโปรเซสเซอร์นี้เราเพียงแต่กดคีย์บอร์ด ก็จะได้แรงดันเอาท์พุทตามที่ต้องการ ความถูกต้องของแรงดันเอาท์พุทจะสูงกว่า REGULATOR ทั่วๆไป

จากการที่กล่าวมาจากข้างแล้วว่าชุดควบคุมเหล่งจ่ายไฟ ด้วยไมโครโปรเซสเซอร์ ยังไม่แพร่หลาย ดังนั้นในการประดิษฐ์ชุดควบคุมเหล่งจ่ายไฟโดยไมโครโปรเซสเซอร์ขึ้นมา จึงมีอุปสรรคบางประการ เช่น การหาข้อมูลอ้างอิง ตลอดจนอุปกรณ์การสร้าง

เนื่องด้วยอุปกรณ์ที่ใช้ในวงจรนั้น จะต้องมีคุณภาพที่ดี พอแต่ดังที่ทราบกันอยู่ว่า ประเทศของเรานั้นเทคโนโลยีทางด้านอิเลคทรอนิคส์นั้นยังไม่เจริญพอทำให้อุปกรณ์บางตัวไม่ได้มาตรฐาน เป็นผลทำให้การทดลองมีความยุ่งยาก เพราะผลที่ได้ไม่เป็นไปตามทฤษฎี ตลอดจนอุปกรณ์บางตัวในท้องตลาดหายากมาก จึงทำให้เกิดความล้าช้า

แต่อย่างไรก็ตาม ชุดควบคุมเหล่งจ่ายไฟด้วยไมโครโปรเซสเซอร์ก็สามารถสร้างขึ้นมาจนสำเร็จ ประสิทธิภาพจัดได้ว่าอยู่ในขั้นที่สามารถยอมรับได้

กิตติกรรมประกาศ

โครงการนี้เป็นผลสำเร็จเรียบร้อยได้ โดยได้รับคำแนะนำปรึกษาและช่วยเหลือจากอาจารย์ที่ปรึกษา ตลอดจนสถานที่วัดทำโครงการของภาควิชาเทคนิคอุตสาหกรรมคหะและวิศวกรรมศาสตร์ ขอขอบคุณ ผ.ศ นิภา สุขุตมตันติ ไว้ ณ ที่นี้ด้วย.

ผู้จัดทำ

เอกสารอ้างอิง

กฤษดา วิศวธีรานนท์และยืน ภู่วรวิริณ ไมโครโปรเซสเซอร์ กรุงเทพมหานคร : บริษัทเอเชียเพรส จำกัด, 2531

Don Lancaster TTL Cookbook

Gayakwad, Ramakant A; " Op - Amps and Linear Integrated Circuits ": Prentice - Hall ; 1988