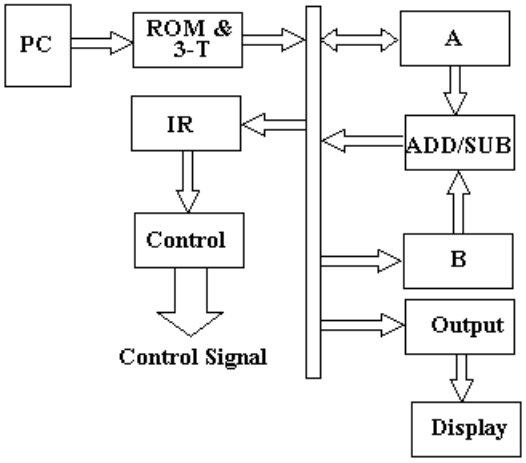
2. SiCo คอมพิวเตอร์อย่างง่าย (Simple Computer)

2.1 บทนำ

เพื่อให้มีความเข้าใจถึงการทำงานของดิจิตอลคอมพิวเตอร์ อันจะนำไปสู่การทำความเข้าใจระบบไมโครโปรเซสเซอร์ ในบทนี้จึงยกตัวอย่างการออกแบบวงจรดิจิตอลคอมพิวเตอร์อย่างง่าย ที่สามารถทำงานได้ แต่มีความซับซ้อนของระบบน้อยเพื่อให้ง่ายต่อการทำความเข้าใจ คอมพิวเตอร์นี้เราเรียกว่า "SiCo" ซึ่งมาจากคำว่า "Simple Computer"

2.2 โครงสร้างของ SiCo

บล็อกไดอะแกรมของ SiCo เป็นตามรูปที่ 2.1 ส่วนประกอบที่สำคัญของระบบประกอบด้วย PC (Program Counter) หน่วยความจำ ROM รีจิสเตอร์ A รีจิสเตอร์ B หน่วยประมวลผล (ALU) รีจิสเตอร์เอาท์พุท รีจิสเตอร์คำสั่ง IR (Instruction Register) และ วงจรควบคุม

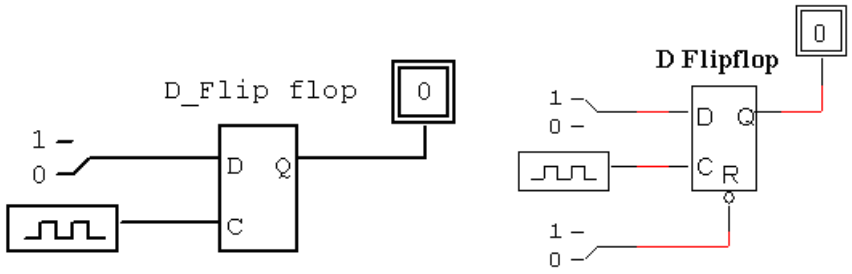


รูปที่ 2.1 บล็อกไดอะแกรมของ SiCo

2.3 รีจิสเตอร์

รีจิสเตอร์ใช้สำหรับเก็บข้อมูลชั่วคราว เพื่อนำข้อมูลนั้นไปประมว ลผลหรือส่งต่อให้หน่วยอื่น ใน SiCo มีการใช้งานรีจิสเตอร์อยู่หลายตัวเช่น รีจิสเตอร์ A รีจิสเตอร์ B และรีจิสเตอร์เอาท์พุท โครงสร้างพื้นฐานของรีจิสเตอร์ก็คือฟลิปฟลอป

2.3.1 D Flipflop

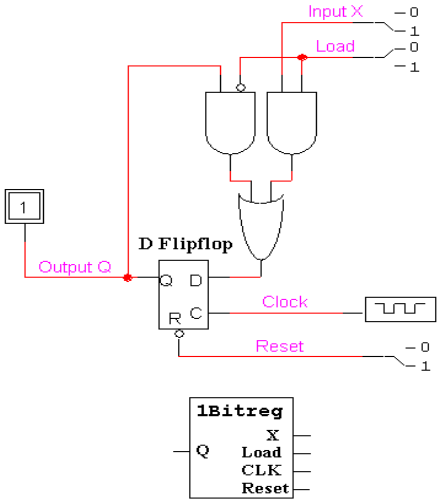


รูปที่ 2.2 ลักษณะของ D Flipflop ทั้งแบบมีสัญญาณรีเซ็ตและแบบไม่มีสัญญาณรีเซ็ต

การทำงานของ ฟลิปฟลอปแบบ D

เอาท์พุท Q เปลี่ยนไปตามค่าของ อินพุท D ทุกครั้งที่มีสัญญาณ CLK (C) มากระตุ้น แต่ถ้าเป็นชนิดที่มีสัญญาณรีเซ็ท (R) เอาท์พุท Q จะเป็นโลจิก 0 ทันทีเมื่อสัญญาณที่ขา R เป็น 0 โดยไม่สนใจสัญญาณที D หรือ C แต่ถ้าสัญญาณ R เป็น 1 เอาท์พุท Q จะเปลี่ยนไปตามค่าของ อินพุท D ทุกครั้งที่มีสัญญาณ CLK (C) มากระตุ้น

2.3.2 รีจิสเตอร์ขนาด 1 บิต (1_Bit Register)



รูปที่ 2.3 ลักษณะของ รีจิสเตอร์ขนาด 1 บิต

การทำงานของ รีจิสเตอร์ขนาด 1 บิต

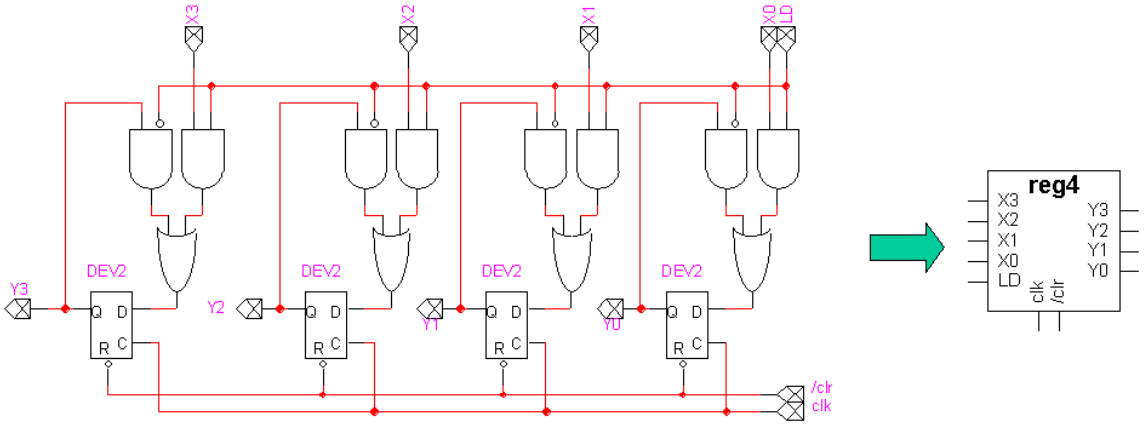
เมื่อ Reset = 0 เอาท์พุท Q = 0

เมื่อ Reset = 1 การทำงานเป็นดังนี้

เมื่อมีสัญญาณนาฬิกา Cถ้า Load = 0 เอาท์พุท Q คงค่าเดิม

ถ้า Load = 1 เอาท์พุท Q = Input X

2.3.3 บัฟเฟอร์รีจิสเตอร์ขนาด 4 บิต (4_Bit Buffer Register)



รูปที่ 2.4 ลักษณะของ รีจิสเตอร์ขนาด 4 บิต

การทำงานของ รีจิสเตอร์ขนาด 4 บิต

ถ้า /CLR = 0 เอาท์พุท Y3Y2Y1Y0 = “0000”

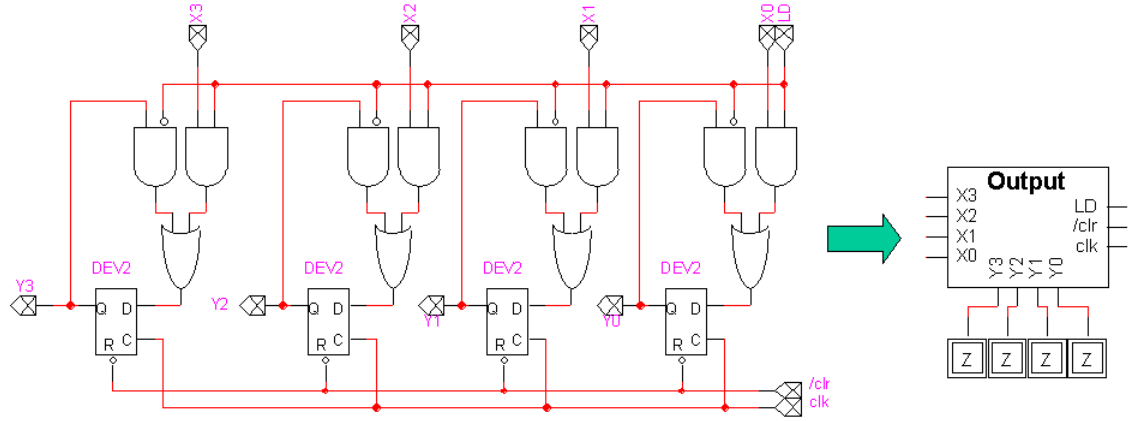
ถ้า /CLR = 1 การทำงานขึ้นอยู่กับ CLK และ LD โดยเมื่อมี CLK มากระตุ้น

- ถ้า Load = 0 Y3 - Y0 จะคงค่าเดิม
- ถ้า Load = 1 Y3 = X3, Y2 = X2, Y1 = X1 และ Y0 = X0

รีจิสเตอร์นี้สามารถนำมาทำเป็น รีจิสเตอร์เอาท์พุทและรีจิสเตอร์ B ได้

รีจิสเตอร์เอาท์พุท (Output Register)

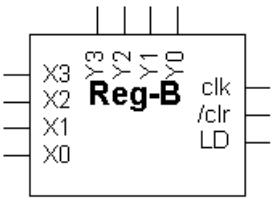
ใช้ Buffer Register ขนาด 4 บิต ทำเป็นรีจิสเตอร์เอาท์พุทสำหรับการแสดงผล



รูปที่ 2.5 ลักษณะของ รีจิสเตอร์เอาท์พุทขนาด 4 บิต

รีจิสเตอร์ B (B Register)

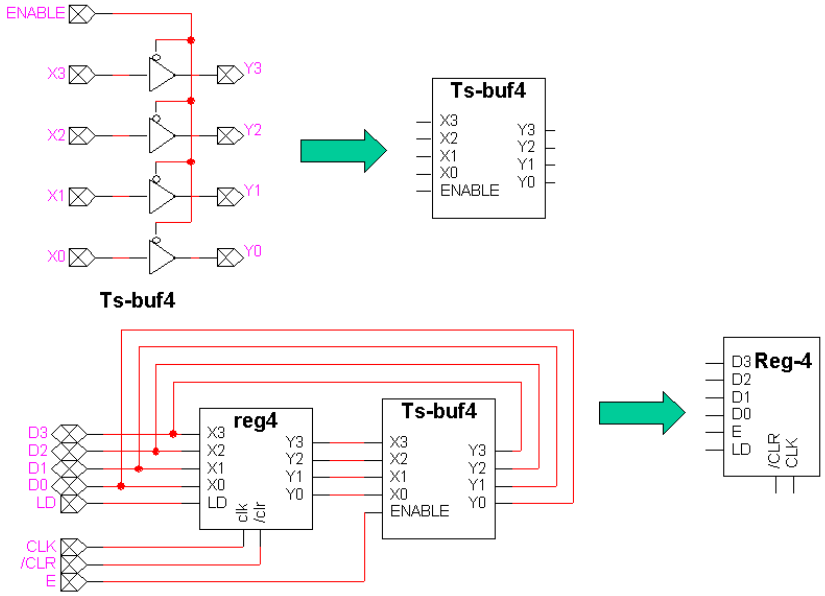
ใช้ Buffer Register ขนาด 4 บิต ทำเป็นรีจิสเตอร์ B เพื่อใช้เก็บข้อมูลทั่วไป



รูปที่ 2.6 ลักษณะของ รีจิสเตอร์เอาท์พุทขนาด 4 บิต

2.3.4 รีจิสเตอร์ 4 บิตพร้อมบัสสองทิศทาง (4_Bit Buffer Register with Bidirectional Bus)

โดยการใช้ บัฟเฟอร์แบบ 3 สถานะ มาประกอบร่วมกับ บัฟเฟอร์รีจิสเตอร์ทำให้ได้เป็นรีจิสเตอร์แบบที่มีบัส 2 ทิศทาง



รูปที่ 2.7 ลักษณะของ รีจิสเตอร์ 4 บิตพร้อมบัสสองทิศทาง

การทำงาน

ถ้า E = 1 สัญญาณเอาท์พุทของ D3, D2, D1 และ D0 จะอยู่ในสถานะ High Impedance ใช้ในสถานะรอรับข้อมูลอินพุทโดย

ถ้า LD = 1 จะรับข้อมูลที่ D3-D0 ไปเก็บไว้ภายในรีจิสเตอร์

ถ้า LD = 0 จะคงค่าข้อมูลเดิมไม่เก็บข้อมูลใหม่

ถ้า E = 0 ข้อมูลที่เก็บอยู่ภายในรีจิสเตอร์จะปรากฏออกทางสัญญาณ D3-D0

สัญญาณ /CLR ใช้สำหรับทำให้ข้อมูลภายในรีจิสเตอร์เป็น 0 ทั้งหมด

โครงสร้างรีจิสเตอร์นี้นำมาทำเป็นรีจิสเตอร์ A เพื่อใช้เป็นรีจิสเตอร์หลักของระบบ

รีจิสเตอร์ A

เป็นรีจิสเตอร์หลักของระบบ การทำงานใดๆ ไม่ว่าจะเป็นการบวก ลบ หรือการส่งผลลัพธ์ออกแสดงผลใช้รีจิสเตอร์ A ทำงานเสมอ ลักษณะของรีจิสเตอร์ A ประกอบด้วย

D3 - D0 เป็นบัส 2 ทิศทาง

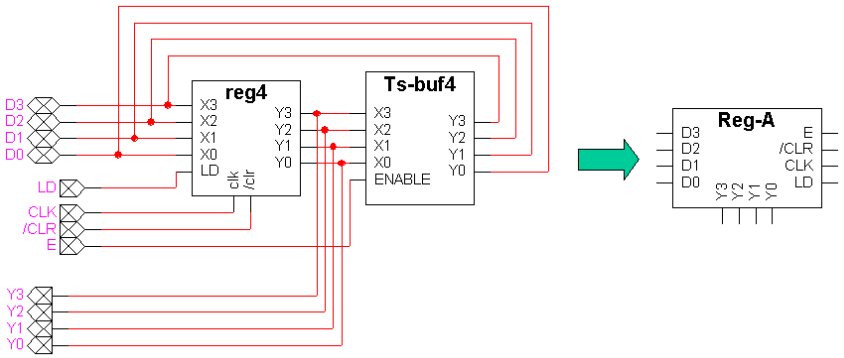
Y3 - Y0 เป็นบัสทิศทางเดียว

CLK เป็นสัญญาณนาฬิกา ใช้กำกับการทำงานของรีจิสเตอร์ เมื่อมีสัญญาณขอบขาขึ้นมา รีจิสเตอร์จึงจะทำงาน ตามการควบคุมของสัญญาณอื่นๆ

LD ใช้ควบคุมการรับข้อมูลจาก D3 - D0 แอคทีฟ 1

E ใช้ควบคุมการส่งข้อมูลที่เก็บอยู่ภายในรีจิสเตอร์ให้ออกมาปรากฏที่ D3-D0 แอคทีฟ 1

/CLR ใช้สำหรับทำให้ข้อมูลภายในรีจิสเตอร์เป็น 0 ทั้งหมด

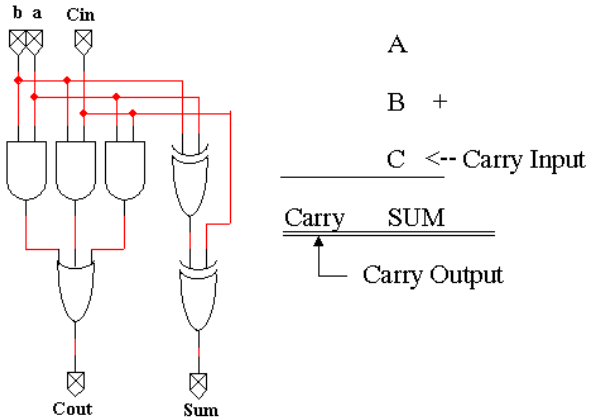


รูปที่ 2.8 บล็อกไดอะแกรมและสัญญลักษณ์ของรีจิสเตอร์ A

2.4 หน่วยประมวลผล (ALU)

หน่วยนี้ทำหน้าที่ในการประมวลผล สำหรับ SiCo ทำได้เพียงการบวกและลบเลขไบนารีขนาด 4 บิตเท่านั้น โครงสร้างประกอบขึ้นจากวงจร Full Adder จำนวน 4 ชุด

วงจรบวก Full Adder



รูปที่ 2.9 โลจิกไดอะแกรมของ Full adder

Arithmetic-Logic Units

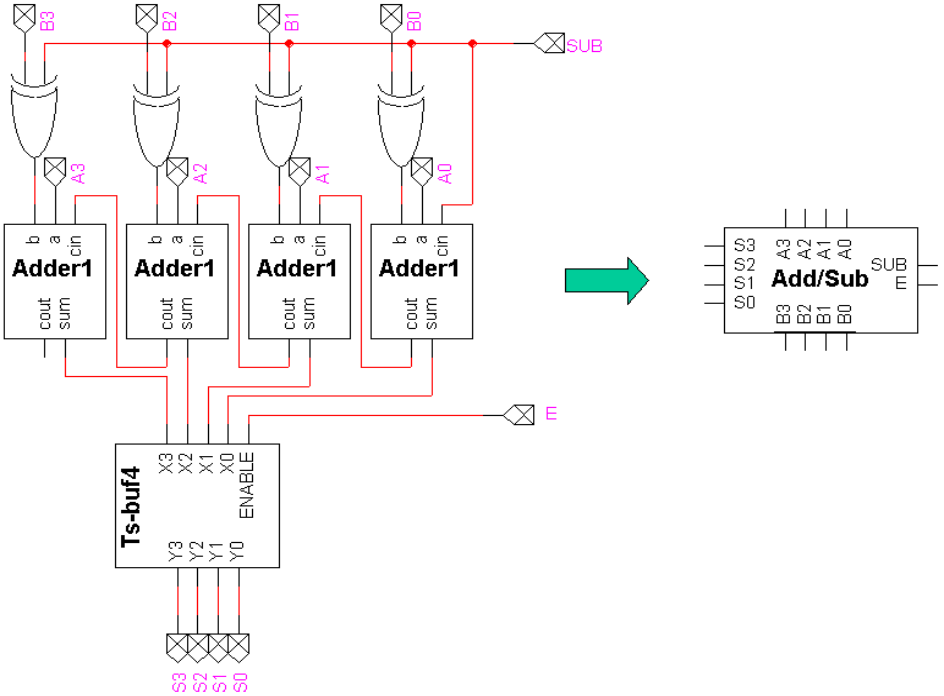
ALU สร้างจากวงจร Full Adder โดยใช้หลักการของ 2's-Complement Adder-subtracter ประกอบเป็นวงจรตามรูปที่ 2.10 และมีการทำงานดังนี้

ถ้า SUB = 0 ได้ S = A + B

ถ้า SUB = 1 ได้ S = A – B

ถ้า E = 0 S มีสัญญาณออกสู่ภายนอก

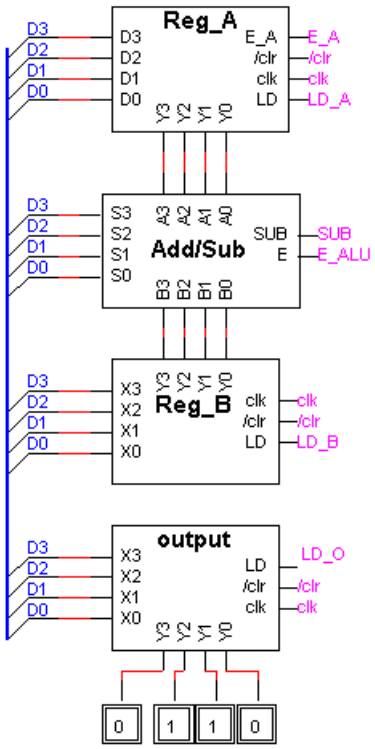
ถ้า E = 1 S เป็น High Impedance



รูปที่ 2.10 โลจิกไดอะแกรมและสัญญลักษณ์ของ ALU

2.5 การต่อรีจิสเตอร์เข้ากับบัส

หลังจากที่มีรีจิสเตอร์ A รีจิสเตอร์ B รีจิสเตอร์เอาท์พุท และ ALU แล้ว ก็นำมาประกอบรวมกันโดยใช้บัส บัสนี้เป็นบัสภายในเป็นของระบบทั้งหมด ทุกๆหน่วยสามารถจะรับและส่งสัญญาณผ่านทางบัสนี้ได้ แต่ต้องผลัดกันใช้ เช่นถ้ารีจิสเตอร์ A ต้องการส่งข้อมูลออกไปให้รีจิสเตอร์เอาท์พุทเพื่อนำออกแสดงผล สัญญาณ E_A ต้องแอคทีฟคือเป็นโลจิก 1 เพื่อให้ข้อมูลในรีจิสเตอร์ A ออกมาที่บัส ขณะเดียวกันสัญญาณ LD_O ก็ต้องแอคทีฟเป็นโลจิก 1 ด้วยเพื่อเก็บข้อมูลที่ปรากฏในบัส เข้าไว้ในรีจิสเตอร์ ส่วนสัญญาณอื่นๆ ณ เวลานี้ต้องไม่แอคทีฟ คือเป็นโลจิกตามตารางที่ 2.1



รูปที่ 2.11 บล็อกไดอะแกรมแสดงการต่อส่วนประกอบต่างเข้าด้วยกันโดยการใช้บัส

ตารางที่ 2.1 แสดงค่าโลจิกที่ให้กับสัญญาณต่างๆเมื่อส่งข้อมูลจาก A ให้แก่ Output

E_A	LD_A	SUB	E_ALU	LD_B	LD_O	/CLR
1	0	0	0	0	1	1

หมายเหตุ ในขณะนี้ต้องมีสัญญาณนาฬิกาอยู่ตลอดเวลา

2.6 Program Counter (PC)

PC เป็นวงจรนับ ใช้สำหรับระบุตำแหน่งของคำสั่งที่จะนำไปทำงานในลำดับถัดไป เมื่อมีการนำคำสั่งออกจากหน่วยความจำแล้ว ค่าของ PC จะเพิ่มค่าไปที่ละ 1 โดยอัตโนมัติ หน้าที่และการทำงานของสัญญาณต่างๆมีดังนี้

Q3 - Q0 เป็นสัญญาณเอาท์พุท

/clr เป็นสัญญาณควบคุมเพื่อทำให้ค่า Q3 - Q0 เป็นโลจิก 0000 โดย

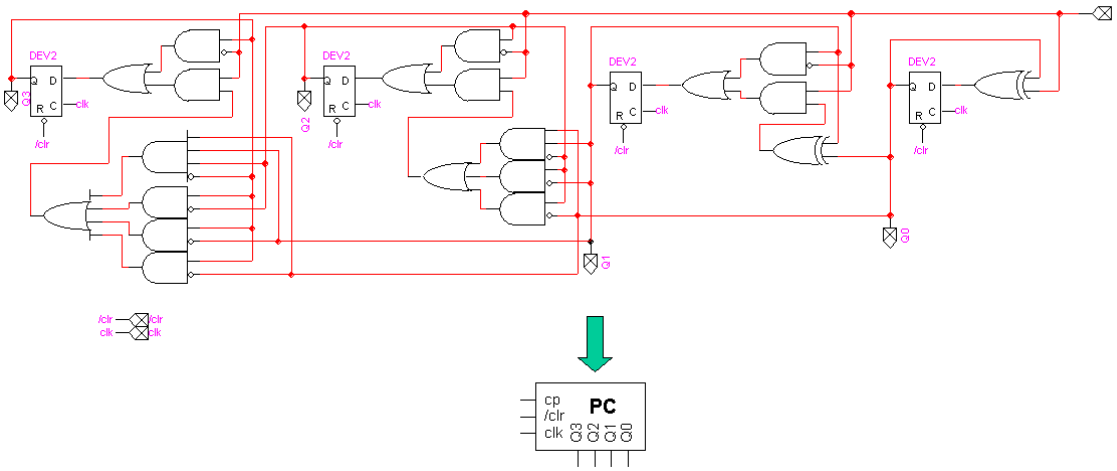
ถ้า /clr = 0 ทำให้ Q3-Q0 = 0000

ถ้า /clr = 1 การทำงานขึ้นอยู่กับ CP และ clk

CP เป็นสัญญาณควบคุมdkioy[Ffp

ถ้า CP = 0 ค่านับคงเดิม

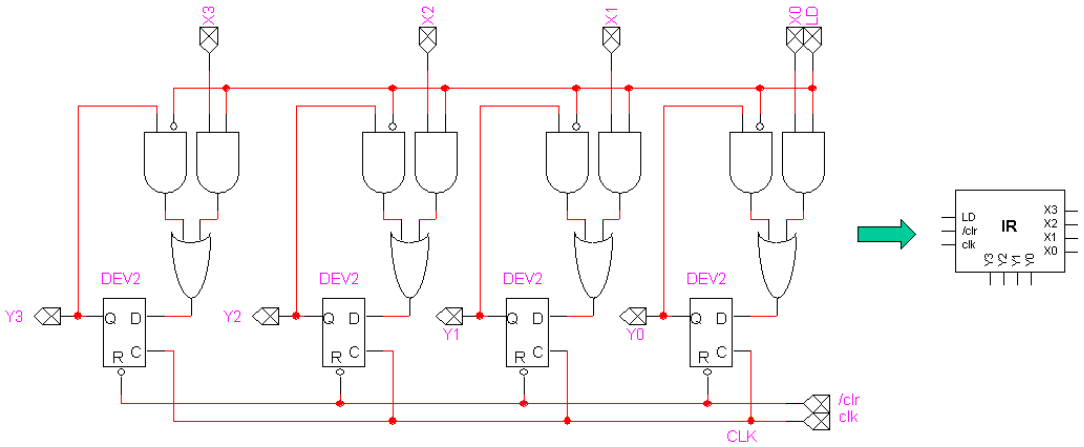
ถ้า CP = 1 นับขึ้น



รูปที่ 2.12 โลจิกไดอะแกรมและสัญญลักษณ์ของ PC

2.7 Instruction Register(IR)

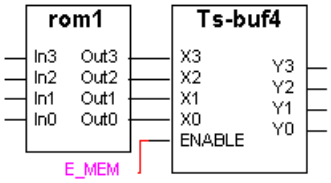
เป็นรีจิสเตอร์ที่ใช้เก็บคำสั่งระหว่างปฏิบัติงาน สร้างจากบัฟเฟอร์รีจิสเตอร์ขนาด 4 บิต



รูปที่ 2.13 โลจิกไดอะแกรมและสัญญลักษณ์ของ IR

2.8 หน่วยความจำ

ใช้สำหรับเก็บโปรแกรมที่ต้องการให้ SiCo หน่วยความจำนี้มีบัสข้อมูลขนาด 4 บิตและบัสแอดเดรสขนาด 4 บิต สามารถเก็บข้อมูลได้ 16 ตำแหน่ง ตำแหน่งละ 4 บิต เนื่องจากต้องต่อหน่วยความจำ



รูปที่ 2.14 โลจิกไดอะแกรมและสัญญลักษณ์ของหน่วยความจำ

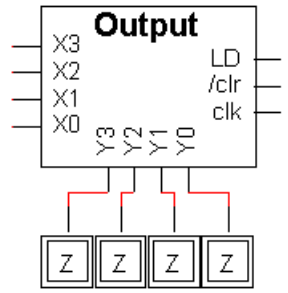
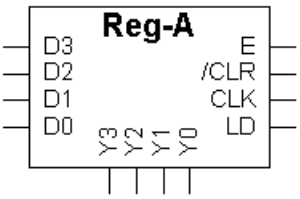
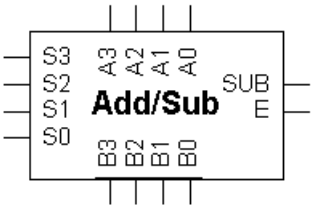
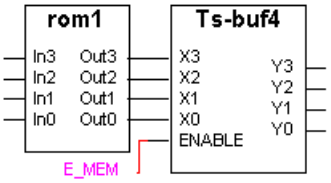
เข้ากับบัสภายใน ดังนั้นสัญญาณเอาท์พุทของหน่วยความจำต้องเป็นแบบ 3 สถานะ จึงใช้บัฟเฟอร์แบบ 3 สถานะมาประกอบ การทำงานเป็นดังนี้

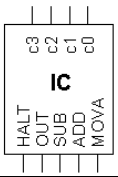
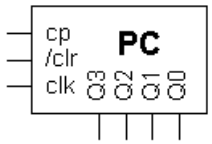
ถ้า E_MEM เป็นโลจิก 1 ข้อมูลจากหน่วยความจำ จะปรากฏที่ Y

ถ้า E_MEM เป็นโลจิก 0 สัญญาณ Y จะเป็น High Impedance

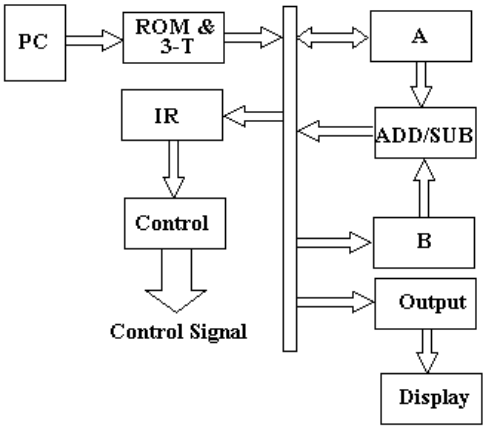
สำหรับข้อมูลในหน่วยความจำนั้นต้องถูกบันทึกไว้ก่อนที่จะนำมาใช้งาน

2.9 สรุปหน้าที่และการทำงานของหน่วยต่างๆของ SiCo

	**รีจิสเตอร์เอาท์พุท (Output Register) รีจิสเตอร์ B และ รีจิสเตอร์คำสั่ง (IR)** อินพุท X0 - X3 เอาท์พุท Y0 – Y3 การทำงาน /clr ทำให้เอาท์พุทเป็น 0 LD = 0 Y คงค่าเดิม LD = 1 Y = X
	**รีจิสเตอร์ A** อินพุท/เอาท์พุท D0 - D3 เป็นสัญญาณ 2 ทิศทาง แบบ 3 สถานะ เอาท์พุท Y0 – Y3 การทำงาน /clr ทำให้เอาท์พุทเป็น 0 LD = 0 Y คงค่าเดิม LD = 1 Y = X E = 0 D0 – D3 ทำหน้าที่เป็นเอาท์พุท E = 1 D0 – D3 ทำหน้าที่เป็นอินพุท ส่วนเอาท์พุทอยู่ในสถานะความต้านทานสูง
ALU 	อินพุท A0 - A3 B0 – B3 เอาท์พุท S0 – S3 เป็นเอาท์พุทแบบ 3 สถานะ การทำงาน SUB = 0 Output = A + B SUB = 1 Output = A - B E = 0 ข้อมูลเอาท์พุทปรากฏที่ S0 – S3 E = 1 S0 – S3อยู่ในสถานะความต้านทานสูง
	**หน่วยความจำ** อินพุท In0 - In3 เอาท์พุท Y0 – Y3 เป็นเอาท์พุทแบบ 3 สถานะ การทำงาน ENABLE = 0 ข้อมูลจาก ROM จะปรากฏที่ Y

	**ถอดรหัสคำสั่ง (Instruction Decoder)** อินพุท C0 - C3 เอาท์พุท HALT, OUT, SUB ADD, HALT แอคทีฟ 1
	**Program Counter (PC)** เอาท์พุท Q0 – Q3 การทำงาน /clr ทำให้เอาท์พุทเป็น 0 CP = 0 ค่านับคงเดิม CP = 1 นับขึ้น

เมื่อนำหน่วยประกอบต่างๆรวมทั้งหน่วยควบคุมมาประกอบเข้าด้วยกันจะมีโครงสร้างดังรูปที่ 2.15



รูปที่ 2.15 บล็อกไดอะแกรมของ SiCo

สำหรับหน่วยควบคุม ซึ่งเป็นหน่วยใหญ่อีกหน่วยหนึ่งนั้น มีความสัมพันธ์กับคำสั่งที่ต้องการให้มีดังนั้นต้องกล่าวถึงคำสั่งเสียก่อน

2.10 คำสั่ง

เนื่องจากต้องการให้ SiCo นั้นง่ายต่อการทำความเข้าใจ จึงกำหนดให้มีคำสั่งเพียง 5 คำสั่งดังนี้

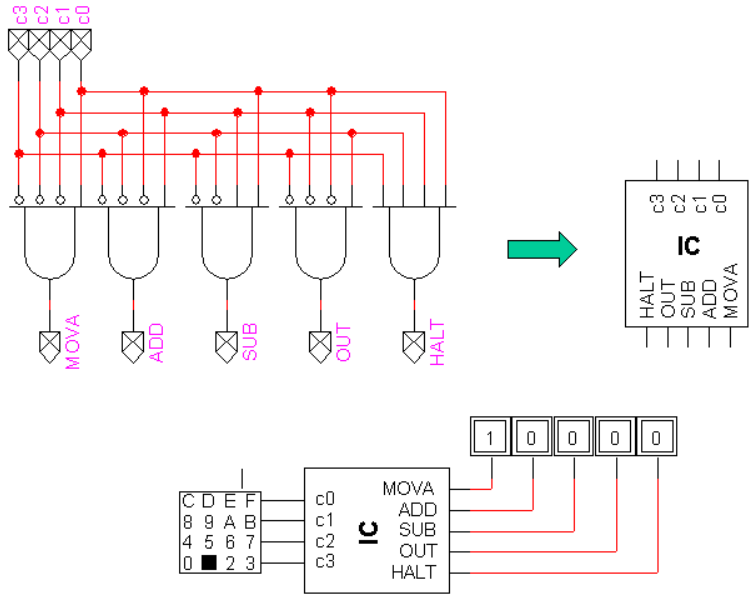
ตารางที่ 2.2 คำสั่งของ SiCo

รหัส	ความหมาย	สัญญลักษณ์การทำงาน	รหัสช่วยจำ
1	เก็บข้อมูลในรีจิสเตอร์ A	A <- n	MOV A,#n
2	บวกข้อมูลในรีจิสเตอร์ A กับข้อมูล n ผลลัพธ์เก็บที่ รีจิสเตอร์ A	A <- A + n	ADD A,#n
3	ลบข้อมูลในรีจิสเตอร์ A ด้วยข้อมูล n ผลลัพธ์เก็บที่ รีจิสเตอร์ A	A <- A - n	SUB A,#n
4	ส่งข้อมูลออกแสดงผล	O/P <- A	OUT
F	หยุดทำงาน		HALT

รหัสคำสั่งหมายถึง คำสั่งที่อยู่ในรูปของเลขฐานสิบหก เช่นถ้าเป็นรหัส 1 จะหมายถึงการนำข้อมูลมาเก็บไว้ในรีจิสเตอร์ A โดยข้อมูลที่มาเก็บ จะอยู่ในหน่วยความจำตำแหน่งถัดจากรหัส 1 ไป

วงจรถอดรหัสคำสั่ง (Instruction Decoder)

เป็นวงจรที่นำเอารหัสคำสั่งต่างๆ ตามตารางที่ 2.2 มาสร้างเป็นสัญญาณควบคุมของแต่ละคำสั่ง เช่น ถ้าคำสั่งที่มีรหัสเป็น 1H (เลขฐานสิบหก) หรือ 0001B (เลขฐานสอง) เข้าสู่วงจรถอดรหัส จะมีเพียงสัญญาณ MOVA เท่านั้นที่เป็นโลจิก 1 นอกนั้นเป็นโลจิก 0 ทั้งหมด



รูปที่ 2.16 โลจิกไดอะแกรมและสัญญลักษณ์ของวงจรถอดรหัสคำสั่ง

การทำงานของคำสั่ง

การที่จะให้เกิดการทำงานต่างๆได้ คำสั่งต่างๆต้องจัดการทำงานเป็นจังหวะ ในที่นี้เรียกว่า T0 ถึง T4 และเพื่อให้ง่ายต่อการทำความเข้าใจ จึงกำหนดรหัสช่วยจำหรือรูปแบบคำสั่งในแบบคำย่อ ดังนี้

ตารางที่ 2.3 การทำงานในแต่ละจังหวะของคำสั่งต่างๆ

รหัส	รหัสช่วยจำ	การทำงาน	T0	T1	T2	T3	T4
1	MOV A,#n	A <- n	IR <- Memory	PC <- PC+1	A <- Memory	PC <- PC+1	
2	ADD A,#n	A <- A + n	IR <- Memory	PC <- PC+1	B <- Memory	PC <- PC+1	A <- (A+B)
3	SUB A,#n	A <- A - n	IR <- Memory	PC <- PC+1	B <- Memory	PC <- PC+1	A <- (A-B)
4	OUT	O/P <- A	IR <- Memory	PC <- PC+1	O/P <- A		
F	HALT		IR <- Memory	PC <- PC+1	STOP		

อธิบายความหมายของตารางที่ 2.3

คำสั่งทุกคำสั่งจะจัดการทำงานเป็น 5 จังหวะ แต่ละจังหวะใช้สัญญาณนาฬิกา 1 ไซเคิล ดังนั้นคำสั่งแต่ละคำสั่งใช้เวลาทำงานเท่ากับ 5 คาบเวลาของสัญญาณนาฬิกา ความหมายของคำสั่งที่เขียนไว้ในตารางที่ 2.3 มีดังนี้

คำสั่งรหัส 1 หมายถึงคำสั่งให้เก็บข้อมูล n ขนาด 4 บิตไว้ที่รีจิสเตอร์ A มี รูปแบบคำสั่ง MOV A,#n มีการทำงานในแต่ละจังหวะดังนี้

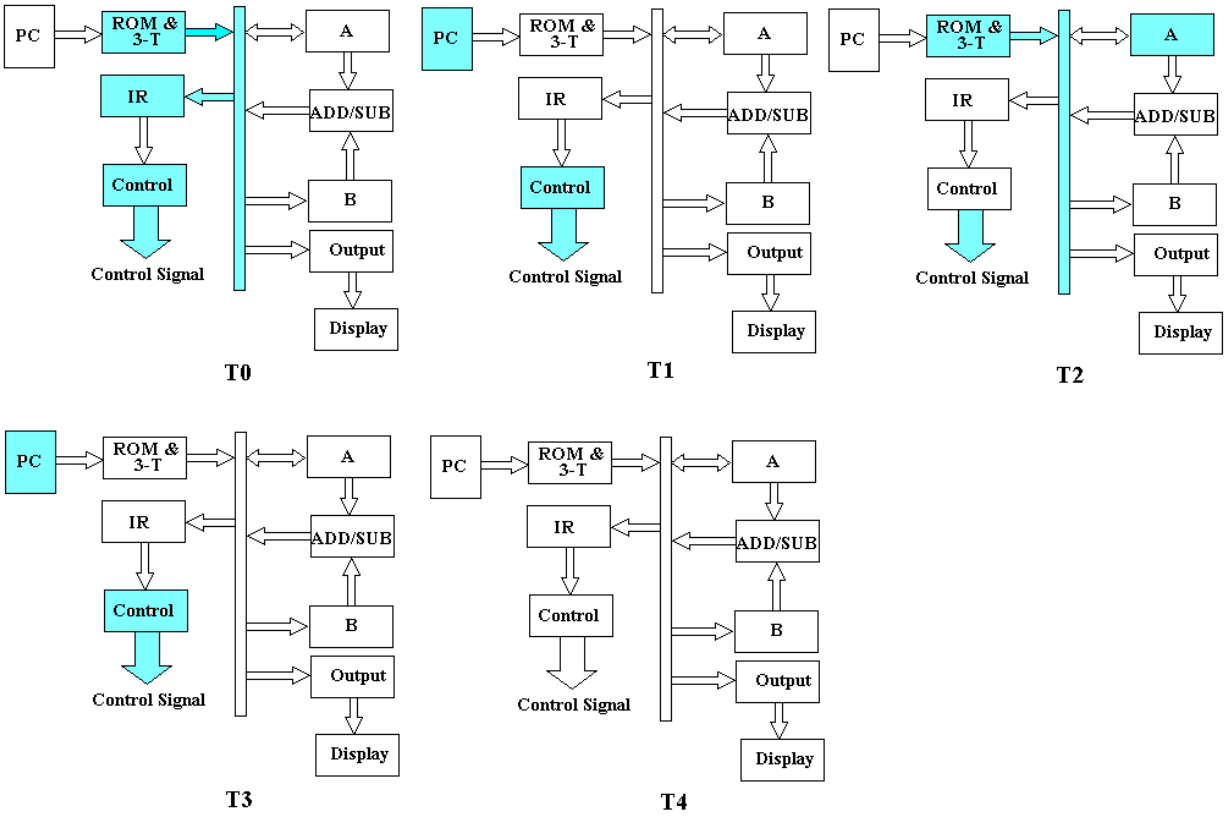
จังหวะแรก T0 IR <- Memory หมายถึง นำคำสั่งที่อยู่ในหน่วยความจำมาเก็บที่รีจิสเตอร์คำสั่ง IR

จังหวะ T1 PC <- PC+1 หมายถึง เพิ่มค่า PC ขึ้น 1

จังหวะ T2 A <- Memory หมายถึง นำข้อมูลที่อยู่ในหน่วยความจำมาเก็บที่รีจิสเตอร์คำสั่ง A

จังหวะ T3 PC <- PC+1 หมายถึง เพิ่มค่า PC ขึ้น 1

จังหวะ T4 หมายถึง ไม่มีการทำงานใดๆ ปล่อยให้สัญญาณนาฬิกาผ่านไป 1 ไซเคิล



รูปที่ 2.17 การทำงานของคำสั่ง MOV A,#n

คำสั่งรหัส 2 หมายถึงคำสั่งให้บวกข้อมูลในรีจิสเตอร์ A เข้ากับข้อมูล n ผลลัพธ์เก็บที่รีจิสเตอร์ A มีรูปแบบคำสั่ง ADD A,#n มีการทำงานในแต่ละจังหวะดังนี้

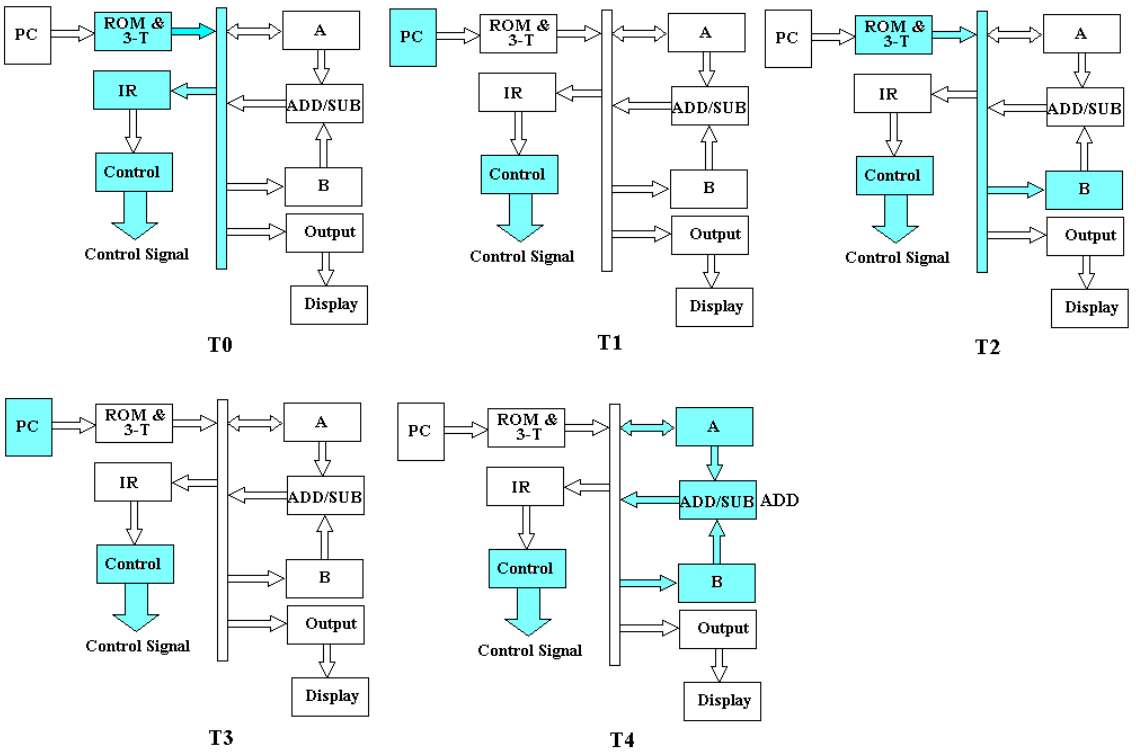
จังหวะแรก T0 IR <- Memory หมายถึง นำคำสั่งที่อยู่ในหน่วยความจำมาเก็บที่รีจิสเตอร์คำสั่ง IR

จังหวะ T1 PC <- PC+1 หมายถึง เพิ่มค่า PC ขึ้น 1

จังหวะ T2 B <- Memory หมายถึง นำข้อมูลที่อยู่ในหน่วยความจำมาเก็บที่รีจิสเตอร์คำสั่ง B

จังหวะ T3 PC <- PC+1 หมายถึง เพิ่มค่า PC ขึ้น 1

จังหวะ T4 A <- A + B หมายถึง บวกข้อมูลในรีจิสเตอร์ A เข้ากับข้อมูลในรีจิสเตอร์ B ผลลัพธ์เก็บที่รีจิสเตอร์ A



รูปที่ 2.18 การทำงานของคำสั่ง ADD A,#n

คำสั่งรหัส 3 หมายถึงคำสั่งให้ลบข้อมูลในรีจิสเตอร์ A ด้วยข้อมูล n ผลลัพธ์เก็บที่รีจิสเตอร์ A มีรูปแบบคำสั่ง SUB A,#n มีการทำงานในแต่ละจังหวะดังนี้

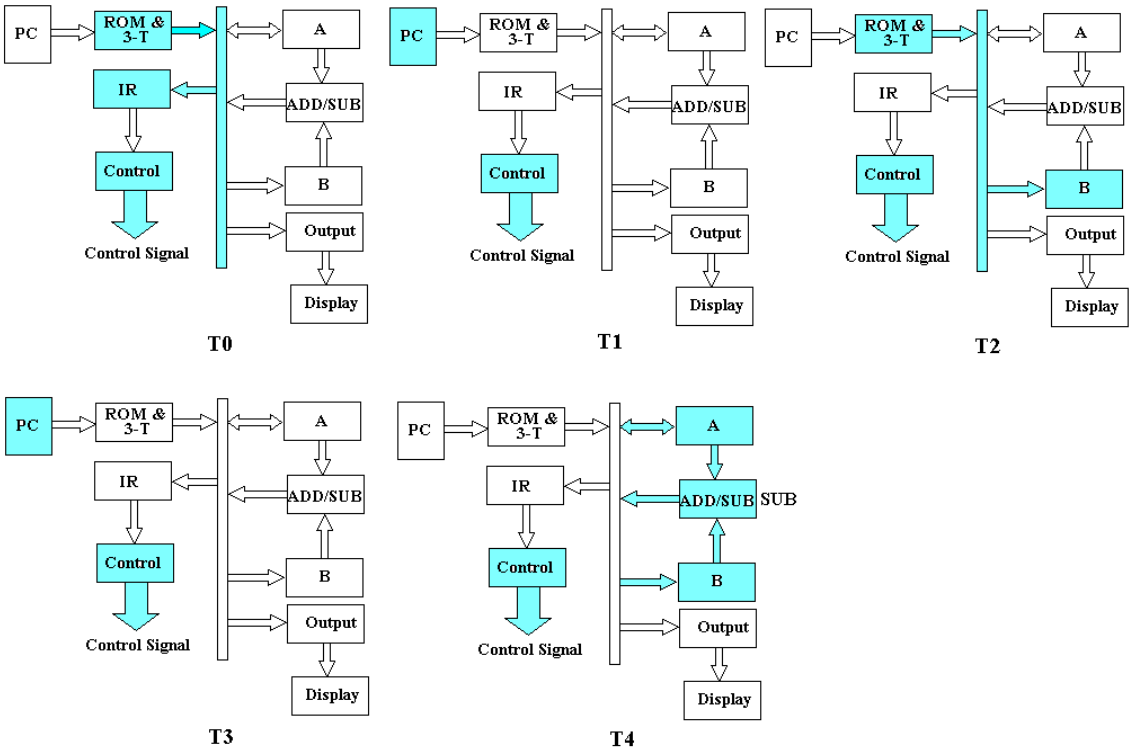
จังหวะแรก T0 IR <- Memory หมายถึง นำคำสั่งที่อยู่ในหน่วยความจำมาเก็บที่รีจิสเตอร์คำสั่ง IR

จังหวะ T1 PC <- PC+1 หมายถึง เพิ่มค่า PC ขึ้น 1

จังหวะ T2 B <- Memory หมายถึง นำข้อมูลที่อยู่ในหน่วยความจำมาเก็บที่รีจิสเตอร์คำสั่ง B

จังหวะ T3 PC <- PC+1 หมายถึง เพิ่มค่า PC ขึ้น 1

จังหวะ T4 A <- A - B หมายถึง ลบข้อมูลในรีจิสเตอร์ A ด้วยข้อมูลในรีจิสเตอร์ B ผลลัพธ์เก็บที่รีจิสเตอร์ A



รูปที่ 2.19 การทำงานของคำสั่ง SUB A,#n

คำสั่งรหัส 4 หมายถึงคำสั่งให้ส่งข้อมูลในรีจิสเตอร์ A ออกไปที่เอาท์พุท มีรูปแบบคำสั่ง OUT มีการทำงานในแต่ละจังหวะดังนี้

จังหวะแรก T0 IR <- Memory หมายถึง นำคำสั่งที่อยู่ในหน่วยความจำมาเก็บที่รีจิสเตอร์คำสั่ง IR

จังหวะ T1 PC <- PC+1 หมายถึง เพิ่มค่า PC ขึ้น 1

จังหวะ T2 O/P <- A หมายถึง นำข้อมูลที่อยู่ในรีจิสเตอร์คำสั่ง A ส่งให้รีจิสเตอร์เอาท์พุท

จังหวะ T3 หมายถึง ไม่มีการทำงานใดๆ ปล่อยให้สัญญาณนาฬิกาผ่านไป 1 ไซเคิล

จังหวะ T4 หมายถึง ไม่มีการทำงานใดๆ ปล่อยให้สัญญาณนาฬิกาผ่านไป 1 ไซเคิล

คำสั่งรหัส 5 หมายถึงคำสั่งให้หยุดการทำงาน มี รูปแบบคำสั่ง HALT มีการทำงานในแต่ละจังหวะดังนี้

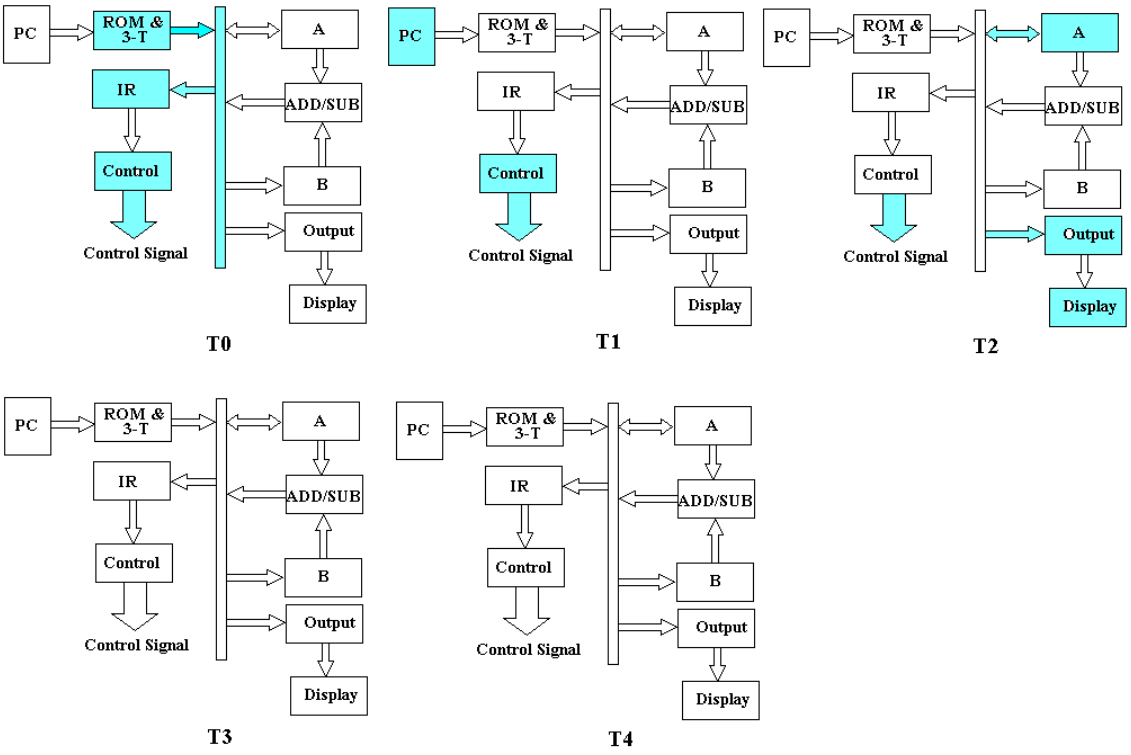
จังหวะแรก T0 IR <- Memory หมายถึง นำคำสั่งที่อยู่ในหน่วยความจำมาเก็บที่รีจิสเตอร์คำสั่ง IR

จังหวะ T1 PC <- PC+1 หมายถึง เพิ่มค่า PC ขึ้น 1

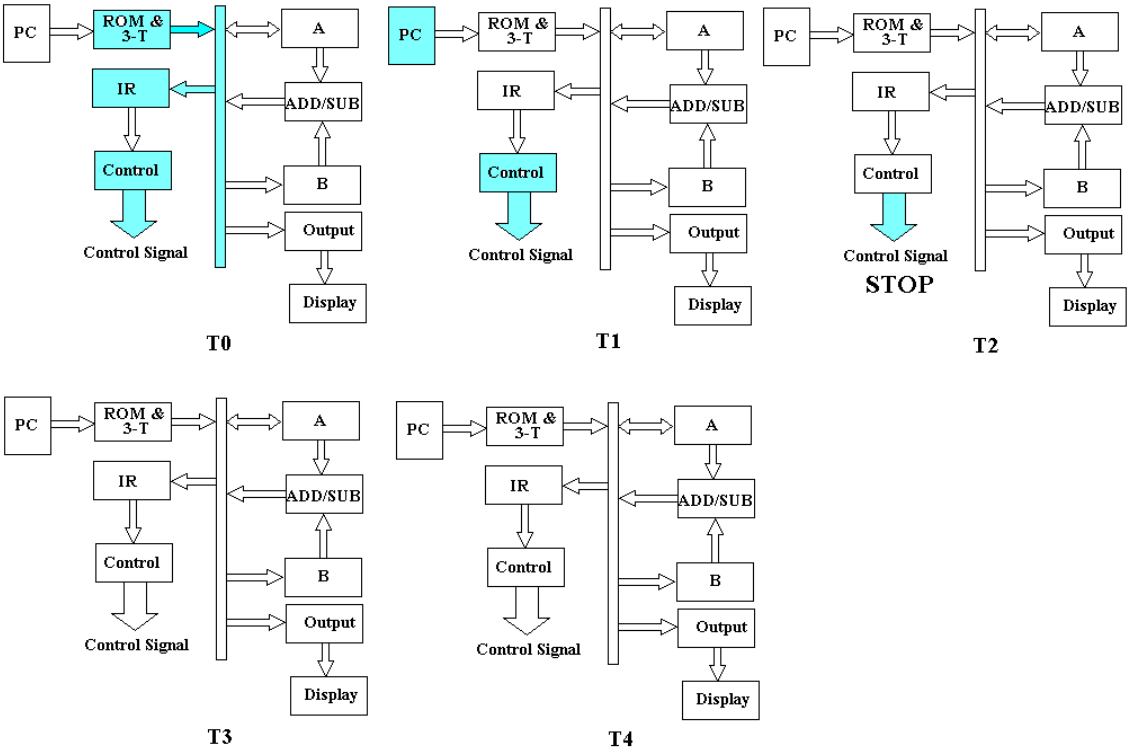
จังหวะ T2 STOP หมายถึง หยุดการทำงาน ในที่นี้ใช้วิธีหยุดจ่ายสัญญาณนาฬิกา

จังหวะ T3 หมายถึง ไม่มีการทำงานใดๆ

จังหวะ T4 หมายถึง ไม่มีการทำงานใดๆ



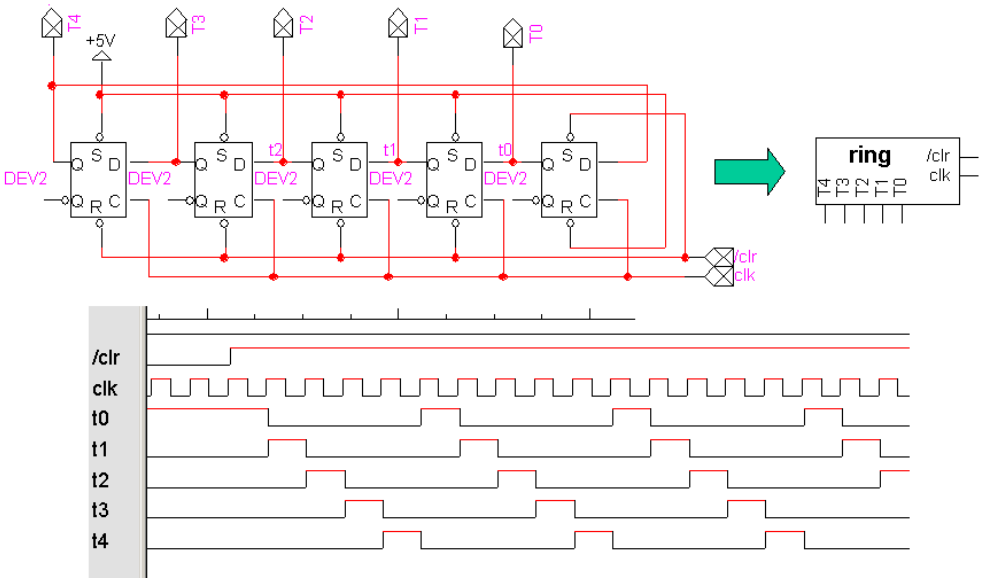
รูปที่ 2.20 การทำงานของคำสั่ง OUT



รูปที่ 2.21 การทำงานของคำสั่ง HALT

2.11 วงจรนับแบบวงแหวน (Ring Counter)

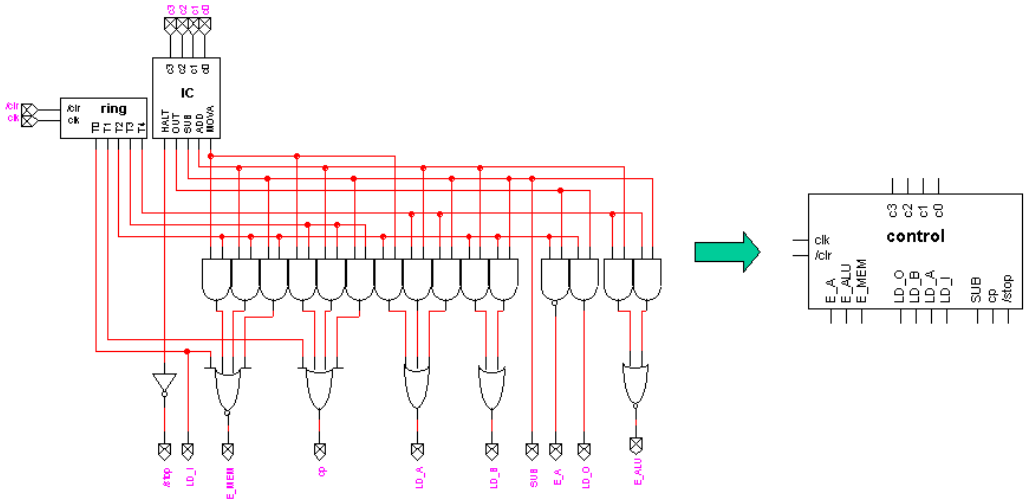
วงจรนับแบบวงแหวนมีลักษณะการทำงานที่ในแต่ละเวลาจะให้สัญญาณเอาท์พุทเป็นโลจิก 1 ได้ครั้งละ 1 บิตเท่านั้น ดังนั้นจึงใช้วงจรนับแบบนี้เป็นหน่วยกำหนดจังหวะการทำงาน โดยถ้าบิตเอาท์พุท T0 เป็น 1 หมายถึงขณะนี้เป็นจังหวะ T0 บิตเอาท์พุทอื่นๆก็เป็นไปในทำนองเดียวกัน



รูปที่ 2.22 โลจิกไดอะแกรมและสัญญลักษณ์ของวงจรนับแบบวงแหวน

2.12 หน่วยควบคุม (Control Unit)

หน่วยควบคุมทำหน้าที่ควบคุมการทำงานของระบบให้เป็นไปตามคำสั่งต่างๆที่ออกแบบไว้ วงจรควบคุมประกอบด้วย วงจรนับแบบวงแหวน วงจรถอดรหัสคำสั่ง และวงจรเกตต่างๆที่ทำหน้าที่ถอดรหัสคำสังและจังหวะการทำงานที่ได้จากวงจรนับแบบวงแหวน แล้วสร้างเป็นสัญญาณควบคุมส่งไปยังหน่วยต่างๆภายในระบบ ตามตารางที่ 2.4



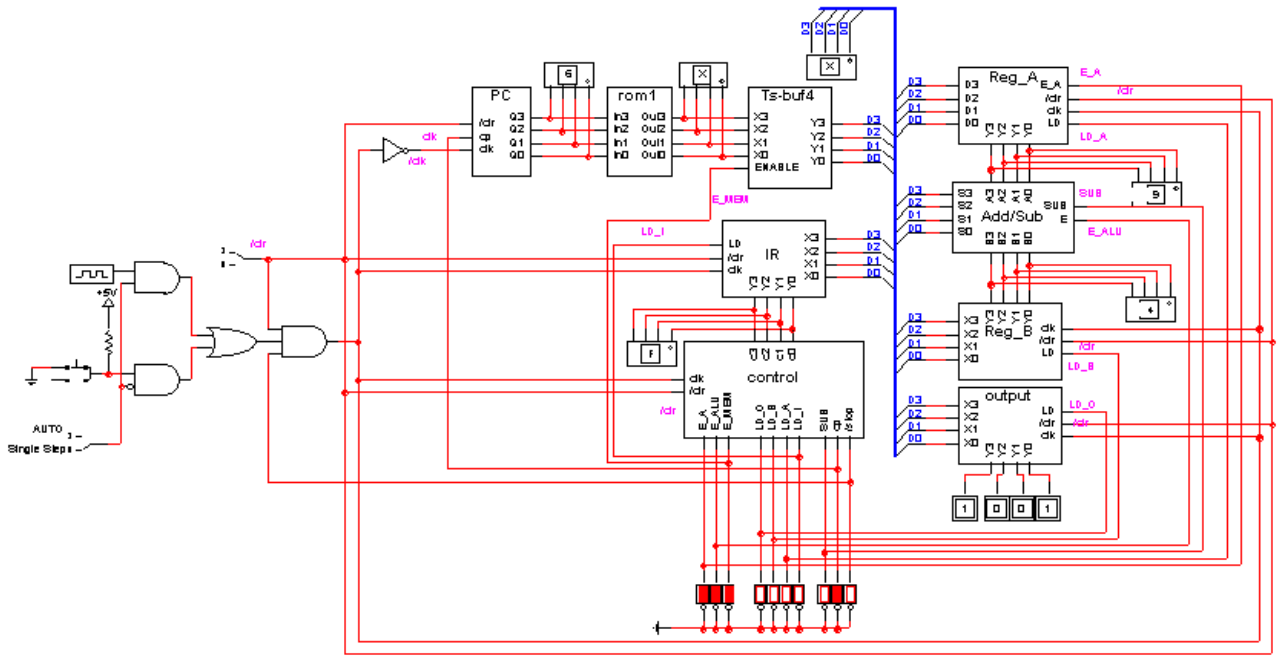
รูปที่ 2.23 โลจิกไดอะแกรมและสัญญลักษณ์ของหน่วยควบคุม

ตารางที่ 2.4 สัญญาณที่ทำงานในแต่ละจังหวะ

		LD_I	LD_A	LD_B	LD_O	E_MEM	E_A	E_ALU	CP	SUB	STOP	ความหมาย
MOV A,#n	T0	1	0	0	0	0	1	1	0	0	1	IR <- Memory
	T1	0	0	0	0	1	1	1	1	0	1	PC <- PC+1
	T2	0	1	0	0	0	1	1	0	0	1	A <- Memory
	T3	0	0	0	0	1	1	1	1	0	1	PC <- PC+1
	T4	0	0	0	0	1	1	1	0	0	1	
ADD A,#n	T0	1	0	0	0	0	1	1	0	0	1	IR <- Memory
	T1	0	0	0	0	1	1	1	1	0	1	PC <- PC+1
	T2	0	0	1	0	0	1	1	0	0	1	B <- Memory
	T3	0	0	0	0	1	1	1	1	0	1	PC <- PC+1
	T4	0	1	0	0	1	1	0	0	0	1	A <- A + B
SUB A,#n	T0	1	0	0	0	0	1	1	0	0	1	IR <- Memory
	T1	0	0	0	0	1	1	1	1	0	1	PC <- PC+1
	T2	0	0	1	0	0	1	1	0	0	1	B <- Memory
	T3	0	0	0	0	1	1	1	1	1	1	PC <- PC+1
	T4	0	1	0	0	1	1	0	0	1	1	A <- A - B
OUT	T0	1	0	0	0	0	1	1	0	0	1	IR <- Memory
	T1	0	0	0	0	1	1	1	1	0	1	PC <- PC+1
	T2	0	0	0	1	1	0	1	0	0	1	O/P <- A
	T3	0	0	0	0	1	1	1	0	0	1	
	T4	0	0	0	0	1	1	1	0	0	1	
HALT	T0	1	0	0	0	0	1	1	0	0	1	IR <- Memory
	T1	0	0	0	0	1	1	1	1	0	1	PC <- PC+1
	T2	0	0	0	0	1	1	1	0	0	0	STOP
	T3	0	0	0	0	1	1	1	0	0	0	
	T4	0	0	0	0	1	1	1	0	0	0	

2.13 สถาปัตยกรรมและบล็อกไดอะแกรมของ SiCo

เมื่อนำหน่วยต่างๆมาประกอบรวมกันได้สถาปัตยกรรมของ SiCo ตามรูปที่ 2.19 และบล็อกไดอะแกรมของระบบแสดงอยู่ในรูปที่ 2.15 การทำงานของระบบทำได้ทั้งแบบอัตโนมัติคือทำตั้งแต่คำสั่งแรกจนสินสุดคำสั่งสุดท้าย หรือทำแบบที่ละคำสั่ง โดยการเลือกวิธีการทำงานด้วยสวิตช์ Auto/ Single Step



รูปที่ 2.24 สถาปัตยกรรม SiCo

ตัวอย่างโปรแกรม

เป็นคำสั่งบวกเลข 5 เข้ากับเลข 4 แล้วนำคำตอบออกแสดงผลที่เอาท์พุท โปรแกรมสามารถเขียนได้ดังนี้

		Mnemonic		
Address	Data	Opcode	Operand	Comment
		ORG	0	
0	1 5	MOV	A,#5	; Set A = 5
2	2 4	ADD	A,#4	; Add A with 4
4	4	OUT		; Display result
5	F	HALT		; Stop

โปรแกรมจัดเก็บลงในหน่วยความจำตามตำแหน่งต่างๆดังนี้

หน่วยความจำ

Address	Data
0	1
1	5
2	2
3	4
4	4
5	F

2.14 Microprogramming

จากที่กล่าวมาแล้วในหัวข้อที่ 2.12 ในหน่วยควบคุมจะใช้วงจรเกตเพื่อสร้างสัญญาณควบคุมต่างๆ การทำลักษณะนี้ไม่มีความยืดหยุ่น การปรับแต่งการทำงานทำได้ยากเพราะว่าต้องแก้ไขวงจรใหม่ อีกวิธีหนึ่งที่นิยมใช้กันมากกว่าคือการใช้ หน่วยความจำ ROM มาแทนวงจรเกต โดยให้บัสแอดเดรสแทนสัญญาณอินพุทของวงจรเกตและ บัสข้อมูลเป็นสัญญาณควบคุมต่างๆ เมื่อต้องการให้สัญญาณควบคุมเป็นอะไรเมื่อสัญญาณอินพุทเป็นอย่างไรก็ใช้การบันทึกเก็บไว้ใน ROM ตัวอย่างเช่น คำสั่ง MOV A,#n จากตารางที่ 2.4 เมื่อจังหวะ T0 สัญญาณควบคุมเป็นดังนี้

		LD_I	LD_A	LD_B	LD_O	E_MEM	E_A	E_ALU	CP	SUB	STOP
MOV A,#n	T0	1	0	0	0	0	1	1	0	0	1

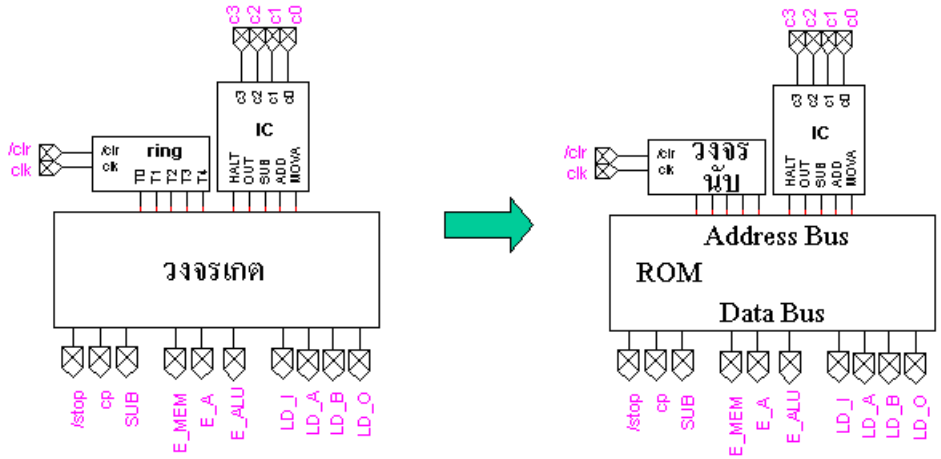
เมื่อเขียนเป็นค่าโลจิกต่างๆได้เป็น

T0	T1	T2	T3	T4	HALT	OUT	SUB	ADD	MOV	LD_I	LD_A	LD_B	LD_O	E_MEM	E_A	E_ALU	CP	SUB	STOP
1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	1

แปลงเป็นสัญญาณสำหรับ ROM

Address Bus										Data Bus									
A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	1

หมายความว่าที่ตำแหน่ง 1000000001B ของ ROM จะต้องบันทึกข้อมูลเป็น 1000011001B การทำเช่นนี้เรียกว่า " Microprogramming" การใช้สัญญาณจาก ROM มาทำหน้าที่แทนวงจรเกตทำให้สามารถแก้ไขและเปลี่ยนแปลงการทำงานของวงจรได้ง่ายขึ้น



รูปที่ 2.25 ลักษณะการใช้ Microprogramming แทนวงจรเกต