บทที่ 4

การออกแบบวงจรรวมจัดการข้อมูลแถวคอย

ดังที่ได้กล่าวถึงหลักการโครงสร้างข้อมูลและอัลกอริธึมของแถวคอยในบทที่ 3 แล้วในบทนี้จะกล่าวถึงการนำโครงสร้างข้อมูล และอัลกอริธึมของแถวคอย มาออกแบบเป็นวงจรอิเล็กทรอนิกส์ ซึ่งวงจรที่จะได้จากการออกแบบนี้ สามารถนำไปจัดสร้างเป็นวงจรที่ใช้อุปกรณ์ย่อย ๆ (Discreate Component) หรือจัดสร้างเป็นวงจรรวมเฉพาะกิจ (Application Specific Integrated Circuit) โดยในบทนี้จะกล่าวถึงการจัดสร้างโดยใช้อุปกรณ์วงจรรวมตระกูล 74 ส่วนการจัดสร้างวงจรรวมอุปกรณ์ที่โปรแกรมได้วงจรรวมเซมิคัสตอมเซลมาตรฐานและวงจรรวมฟูลคัสตอม จะได้กล่าวถึงในบทที่ 5 ต่อไป

1. <u>การออกแบบวงจรอิเล็กทรอนิกส์จัดการแถวคอย</u>

ส่วนประกอบของการจัดการแถวคอยที่จะออกแบบสามารถแบ่งออกได้เป็น 2 ส่วน ได้แก่

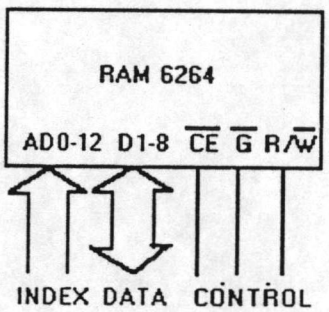
- โครงสร้างข้อมูล
- อัลกอริธึม

ส่วนประกอบที่เป็นโครงสร้างข้อมูลประกอบด้วย LIST, HEAD และ TAIL ส่วนที่เป็นอัลกอริธึมได้แก่ฟังก์ชั่น Q_Empty, Q_Full, Q_Insert, Q_Remove และ Q_Init และภายใน Q_Insert และ Q_Remove ยังมีอัลกอริธึมย่อยคือ Next_Index

การกำหนดและออกแบบโครงสร้างข้อมูล LIST นั้นจะมีผลกับ HEAD และ TAIL ซึ่งต้องทราบจำนวนหรือขนาดของ List ว่ามีส่วนประกอบเป็นจำนวน element เท่าใด ซึ่ง HEAD และ TAIL ที่ใช้เป็นตัวชี้ (Index) ของอาจารย์จะต้องชี้จำนวน element ได้เพียงพอ

1.1 <u>โครงสร้างข้อมูล LIST</u> การสร้างข้อมูล LIST ที่เป็นอาเรย์จะใช้หน่วยความจำ Random Access Memory หรือ RAM ในโครงการนี้จะใช้อุปกรณ์ RAM ที่เป็น Static-RAM ขนาดของแต่ละ element ที่จะใช้จัดเก็บข้อมูลเป็น 8 บิต เนื่องจากข้อมูลที่ใช้จัดเก็บเป็นข้อมูลชนิด CHAR ใช้จัดเก็บอักษรตามมาตรฐานรหัส ASCII จำนวนของ element ของ List นี้ มักจะมีจำนวนเป็น exponential ของ 2 เพื่อให้การอ้างถึงตำแหน่งต่าง ๆ ใน List นั้นเต็มพอดีกับจำนวนบิตของตัวอ้าง เช่น จำนวน element เป็น 1024 ไบต์ (1 กิโลไบต์) ใช้ตัว

รหัส ASCII จำนวนของ element ของ List นี้ มักจะมีจำนวนเป็น exponential ของ 2 เพื่อให้การอ้างถึงตำแหน่งต่าง ๆ ใน List นั้นเต็มพอดีกับจำนวนบิตของตัวอ้าง เช่น จำนวน element เป็น 1024 ไบต์ (1 กิโลไบต์) ใช้ตัวอ้างตำแหน่งที่มีขนาด 10บิต (2^{10}= 1024) ในการวิจัยนี้จะใช้หน่วยความจำขนาด 8192 x 8 บิต ใช้อุปกรณ์หน่วยความจำหมายเลข 6264 จะต้องใช้ตัวแปรที่ใช้อ้างถึงตำแหน่ง element หรือ Index ที่อ้างถึงได้ตั้งแต่ 0-8191 เป็นตัวแปรขนาด 13 บิต ผังรูปภาพของหน่วยความจำที่นำมาใช้เป็น List แสดงดังภาพ 4.1



รูปที่ 4.1 สัญญาณต่างๆ ของอุปกรณ์หน่วยความจำ

กลุ่มของสัญญาณติดต่อแบ่งออกเป็น Index เป็นกลุ่มสัญญาณที่ใช้อ้างถึงตำแหน่ง element ที่ต้องการใช้งานโดยปกติแล้วสำหรับหน่วยความจำเราจะเรียก Index ว่า address กลุ่มสัญญาณ Data เป็นกลุ่มที่ใช้ส่งข้อมูลเข้าสู่หรือรับข้อมูลที่ได้จากหน่วยความจำจะใช้สายสัญญาณร่วมกัน แต่จะใช้ต่างโอกาสกัน สัญญาณในกลุ่ม Control เป็นกลุ่มที่ใช้ควบคุมการทำงานของหน่วยความจำแบ่งเป็น

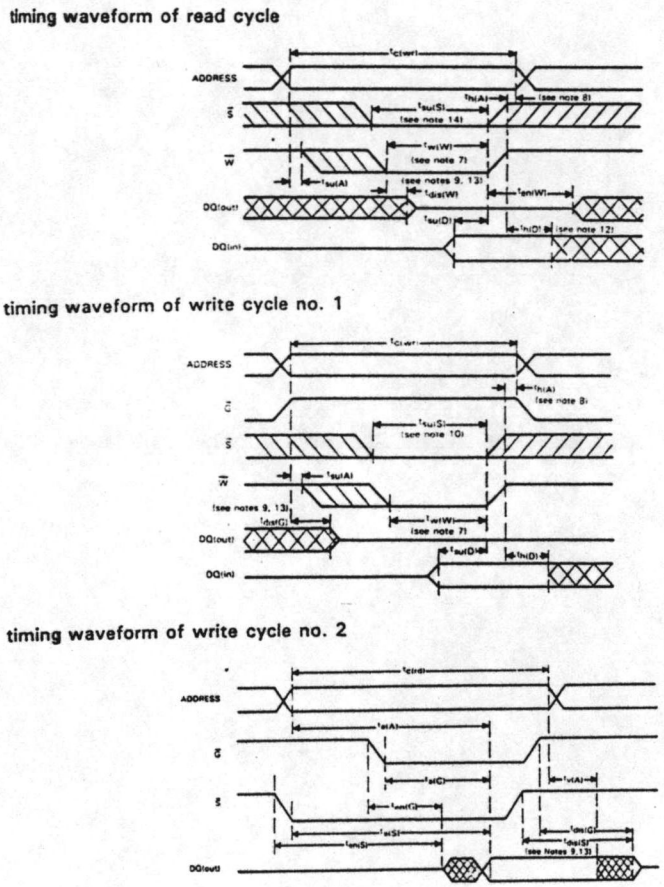
1) CE หรือ chip Enable เป็นสัญญาณที่สั่งการให้อุปกรณ์หน่วยความจำพร้อมที่จะรับคำสั่งอื่น ๆ หากเราให้สัญญาณแก่ CE เป็นลอจิก 0 จะทำให้อุปกรณ์หน่วยความจำนี้พร้อมรับคำสั่งและทำงานแต่หากให้สัญญาณเป็นลอจิก 1 จะทำให้อุปกรณ์หน่วยความจำนี้เข้าสู่สภาวะ Standby ซึ่งจะใช้กระแสไฟฟ้าเลี้ยงภายในต่ำกว่าปกติ แต่จะไม่รับคำสั่งในการอ่านหรือบันทึกข้อมูล

2) G หรือ Output Enable เป็นสัญญาณที่ใช้เปิด-ปิด ข้อมูลจากภายในหน่วยความจำให้ออกมายังสายสัญญาณ Data หรือไม่หากเราให้สัญญาณลอจิก 0 ก็จะเปิดให้สัญญาณจากหน่วยความจำออกมายัง DATA (เว้นแต่เมื่อสัญญาณ R/W เป็น 0) แต่หากเราให้สัญญาณลอจิก 1 เป็นการปิดไม่ให้ข้อมูลจากภายในหน่วยความจำออกมายัง Data

3) R/W เป็นสัญญาณที่สั่งการให้หน่วยความจำอยู่ในสภาวะอ่านหรือบันทึกข้อมูล หากเราให้สัญญาณลอจิก 1 หน่วยความจำจะอยู่ในสภาวะการอ่านข้อมูลซึ่งหากเราให้สัญญาณ Output enable

เป็นลอจิก 0 ด้วยข้อมูลตำแหน่งที่ถูกอ้างด้วย index ก็จะแสดงออกมาที่สัญญาณ Data หากเราให้สัญญาณลอจิก 0 หน่วยความจำจะอยู่ในสภาวะการบันทึกโดยการนำข้อมูลจากสัญญาณ Data บันทึกลงในหน่วยความจำตำแหน่งที่ถูกอ้างถึงอยู่ด้วย Index

ในภาพที่ 4.2 แสดงถึงผังเวลาในการทำงานของอุปกรณ์หน่วยความจำหมายเลข 6264

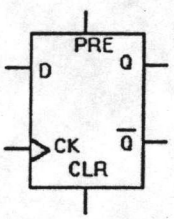


รูปที่ 4.2 ผังเวลาการทำงานของหน่วยความจำ 6264

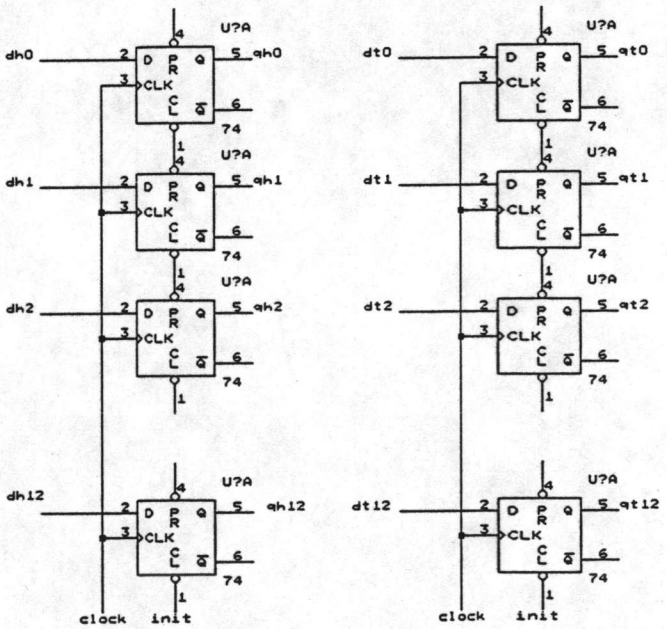
1.2 <u>โครงสร้างข้อมูล HEAD และ TAIL</u> ตัวแปร HEAD และ TAIL ที่ใช้เป็นตัวแปรสำหรับชี้ตำแหน่ง element ในลิสต์คือตัวแปร Index เป็นตัวแปรที่มีชนิดข้อมูลเป็นตัวเลขจำนวนเต็ม (integer) ที่สามารถเก็บบันทึกข้อมูลตั้งแต่ 0 ถึงจำนวน element ของ LIST-1 ในการวิจัยนี้ตัวแปร HEAD และ TAIL เป็นตัวแปรขนาด 13 บิต อ้างถึงตำแหน่งได้ตั้งแต่ 0 ถึง 8191 เราจะสร้างตัวแปรโดยใช้อุปกรณ์ Register แทนแต่ละบิตของข้อมูล คุณลักษณะของ Register ดังกล่าวเป็นอุปกรณ์ที่ทำงานตามจังหวะสัญญาณนาฬิกา (Clock) ที่กำกับ จะรับข้อมูลจากสัญญาณ D และให้ output ออกทาง Q เราสามารถนำ register ดังกล่าวมาใช้ขนานกันให้มีจำนวนบิตตามต้องการในรูปที่ 4.3 แสดงสัญญลักษณ์ของ Register หรือเรียกโดยทั่วไปว่า D-Flip Flop และรูปที่ 4.4 แสดงการจัดโครงสร้างข้อมูลตัวแปร HEAD และ TAIL

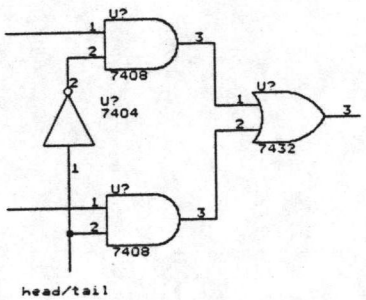
นอกเหนือจากตัวแปร HEAD และ TAIL เองแล้วยังต้องทำงานออกแบบการนำตัวแปรนี้ไปใช้อ้างถึงตำแหน่ง element ในตัวแปรอาเรย์ ซึ่งเป็นหน่วยความจำด้วย ซึ่งการอ้างถึงในบางขณะจะใช้ตัวแปร HEAD ในขณะที่บางขณะอาจใช้ตัวแปร TAIL ดังนั้นจะต้องมีวงจรเลือกนำข้อมูลจาก HEAD หรือ TAIL มาใช้ด้วย ลักณะวงจรดังกล่าวอาจเป็นวงจรมัลติเพลกเซอร์ ดังภาพ 4.5 หรือใช้วงจรบัฟเฟอร์ที่มี 3 สถานะ(Tri-State Buffer) ดังรูป 4.6 ซึ่งในโครงการนี้จะใช้บัฟเฟอร์ 3 สถานะ เนื่องจากใช้อุปกรณ์น้อยชิ้นกว่า)

รูปที่ 4.3 อุปกรณ์ D-Flip Flop ที่ใช้สร้างตัวแปร



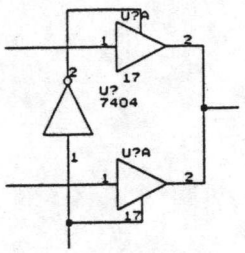
รูปที่ 4.4 การจัดโครงสร้างตัวแปรอินเดกซ์





HEAD/TAIL	OUT
0	HEAD
1	TAIL

รูปที่ 4.5 วงจรมัลติเพลกเซอร์ที่ใช้เลือกอินเดกซ์



HEAD/TAIL	OUT
0	HEAD
1	TAIL

รูปที่ 4.6 วงจรบัฟเฟอร์ 3 สถานะที่ใช้เลือกอินเดกซ์

1.3 อัลกอริธึม Q_Empty และ Q_Full จากสรุปโครงสร้างและอัลกอริธึมของการจัดการแถวคอยในหัวข้อ 3 บทที่ 3 ได้สรุปการตรวจสอบ Q_Empty และ Q_Full เป็นฟังก์ชั่นได้คือ

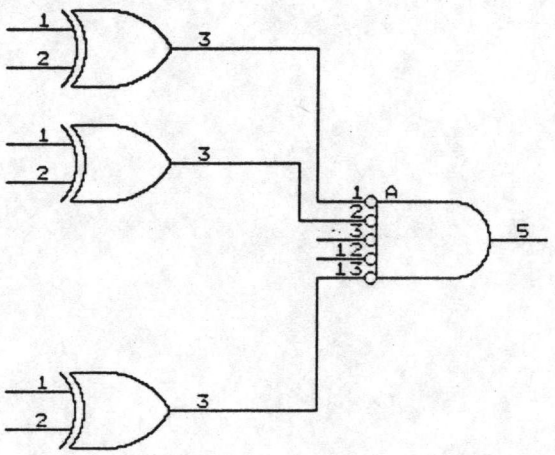
Q_EMPTN := Q.HEAD = Q.TAIL ;

และ

Q_FULL := Next_Index (Q.TAIL) = Q.HEAD ;

ดังนั้นในการจัดสร้างวงจรแทนอัลกอริธึมของการตรวจสอบ Q_Empty และ Q_Full นั้นจะต้องสร้างวงจรสำหรับการเปรียบเทียบข้อมูล 2 ชุด มีค่าเท่ากันหรือไม่ และอัลกอริธึม NEXT_INDEX เป็นอัลกอริธึมที่ใช้คำนวณค่า INDEX ถัดไป ซึ่งจะกล่าวถึง NEXT_INDEX ในหัวข้อ 4.1.4 ต่อไป ส่วนการสร้างวงจรเพื่อเปรียบเทียบข้อมูล 2 ค่า เท่ากันหรือไม่นั้นสามารถทำได้ Exclusive-OR gate เพื่อเปรียบเทียบข้อมูลแต่ละบิตและใช้ AND gate เพื่อตรวจสอบว่าการเปรียบเทียบทุกบิตได้ค่าเท่ากัน ดังรูปที่ 4.7 แสดงการนำ Exclusive-OR gate มาใช้เปรียบเทียบข้อมูลแต่ละบิตและใช้ AND gate ตรวจสอบว่าการเปรียบเทียบทุกบิตเท่ากันหรือไม่ซึ่งการใช้ Exclusive-OR นั้น หากข้อมูลที่นำมาเปรียบเทียบ 2 ค่า ตรงกันจะให้ลอจิกเป็น 0 หากไม่ตรงกันจะได้ลอจิก 1 ดังนั้นการนำไป AND กันนั้นอินพุทของการ AND จะต้องเป็น Inverting Input

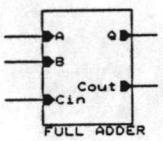
จากทฤษฎีของเดอมอแกน (Demorgan) เราสามารถแทน $C_1 \cdot C_2 \cdot C_3 \ldots C_n$ ได้ด้วย $C_1 + C_2 + C_3 + \ldots + C_n$ ดังนั้นอุปกรณ์ที่ใช้ในการ AND เพื่อตรวจสอบว่าการเปรียบเทียบทุกบิตได้ตรงกันก็สามารถใช้ NOR gate แทนได้ในกรณีที่จะต้องเลือกตัวอุปกรณ์ที่ใช้จัดสร้าง



รูปที่ 4.7 วงจรเปรียบเทียบข้อมูลมีค่าเท่ากันโดยใช้ Exclusive-OR และ AND gate

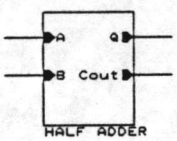
1.4 อัลกอริธึม NEXT_INDEX ภายในอัลกอริธึมต่าง ๆ ของการจัดการแถวคอยมีอัลกอริธึมย่อยตัวหนึ่งที่เป็นส่วนประกอบซึ่งถูกใช้บ่อยครั้งในการจัดการแถวคอยได้แก่อัลกอริธึม NEXT_INDEX ซึ่งจะให้ค่าตำแหน่งอินเดกซ์ ถัดไปของตัวแปร HEAD หรือ TAIL เมื่อเราใช้แถวคอยแบบวงแหวนการหา INDEX ค่าถัดไปนั้นจะต้องวนกลับไปยังตำแหน่งแรก หากค่าอินเด็กซ์ปัจจุบันเป็นค่าสูงสุดในทางซอฟต์แวร์เราสามารถใช้เครื่องหมายการคำนวณหาเศษจากการหาร (modulo) เพื่อจัดการให้ค่าอินเดกซ์เป็นแบบวงแหวนดังกล่าวข้างต้นได้ แต่ในกรณีที่เราออกแบบนี้ได้เลือกขนาดของอาเรย์กับขนาดของตัวแปร HEAD และ TAIL ที่ใช้เป็น INDEX ให้สอดคล้องกัน เมื่อเพิ่มค่าตัวแปร INDEX ต่อจากค่าสูงสุดแล้วค่าที่ได้รับจะกลับเป็นค่า 0 โดยอัตโนมัติดังนั้นในการจัดสร้างเป็นวงจรเราจึงไม่จำเป็นจะต้องใช้วิธีการพิเศษใดอีก แต่สิ่งที่จะต้องออกแบบก็คือวงจรที่ให้ค่าปัจจุบันบวกหนึ่ง (INDEX ปัจจุบัน +1) ซึ่งวงจรสำหรับการบวกค่าเราสามารถใช้วงจรบวกเลข (Adder) ได้โดยวงจรบวกเลขมี 2 ประเภทได้แก่ Half Adder และFull Adder ซึ่งมีข้อแตกต่างกันที่ Full Adder นั้น สามารถจะทดจากหลักก่อนหน้าไปยังหลักถัดไปได้ ในภาพ 4.8 เป็นภาพแสดงสัญญลักษณ์และการทำงานของ Full Adder และ Half Adder

เนื่องจากวงจรที่เราจะจัดสร้างนั้นเราจะใช้เพียงบวกด้วย 1 ซึ่งหากคิดเป็นเลขฐานสอง คือ 0000000000001_2 (ใช้ INDEX ขนาด 13 บิต) แต่ละบิตจะบวกด้วย 0 ยกเว้นบิตล่างสุดที่จะบวกด้วย 1 แต่การบวกจะต้องมี Carry เพื่อทดไปยังหลักถัดไป เนื่องจากการบวกด้วย 0 นั้นอาจได้ผลเท่ากับการบวกกับตัวทดเท่านั้น ดังนั้นเราจะใช้ Half Adder โดยใช้ Input B ของ Adder มาใช้แทน Carry in และบิตล่างสุดนั้น Input B จะให้เป็น 1 เพื่อแสดงการบวกกับ 1 (บิตล่างสุดไม่ต้องใช้ Carry In เนื่องจากไม่ต้องมีการทดลองหลักก่อนหน้า) ภาพ 4.9 แสดงการใช้ Adder เป็น NEXT_INDEX



A	B	Cin	Q	Cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

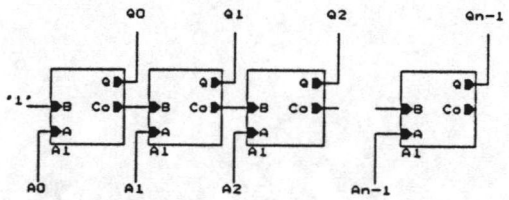
(ก)



A	B	Q	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

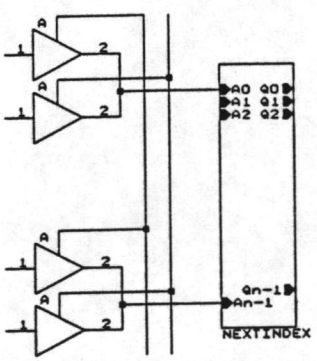
(ข)

รูปที่ 4.8 วงจรบวกเลข (ก) Half Adder (ข) Full Adder

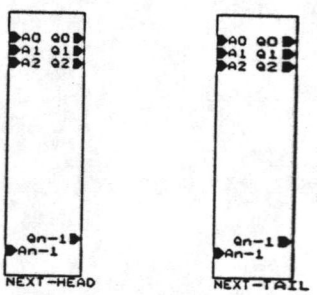


รูปที่ 4.9 วงจร NEXT_INDEX ซึ่งสร้างจากวงจรบวกเลข Half Adder

เนื่องจากอัลกอริธึมของ NEXT_INDEX นี้ถูกนำไปใช้กับ HEAD และ TAIL หากเราสร้างวงจร NEXT_INDEX เพียงวงจรเดียวจะต้องมีวงจรเลือกการนำ HEAD หรือ TAIL เข้าไปยัง INPUT ของวงจร NEXT_INDEX ดังรูป 4.10 โดยการใช้บัฟเฟอร์ 3 สถานะเป็นตัวเลือกโดยจะมีข้อจำกัดจะต้องไม่มีการใช้ NEXT_INDEX พร้อมกันระหว่างตัวแปร HEAD และ TAIL และจะต้องจัดสัญญาณเพื่อเลือก HEAD หรือ TAIL เข้าไปยัง NEXT_INDEX ซึ่งจะต้องสอดคล้องกับการทำงานตามจังหวะขอบขาขึ้นหรือลงของสัญญาณนาฬิกาของตัวแปร HEAD และ TAIL ซึ่งเราใช้ D-FlipFlop ทำหน้าที่เป็น Register อยู่ (หัวข้อ 4.1.2) หากไม่ใช้วิธีการเลือกแบบนี้ก็จะต้องจัดสร้างวงจร NEXT_INDEX จำนวน 2 ชุด สำหรับตัวแปร HEAD และตัวแปร TAIL (หากมีการใช้ NEXT_INDEX โดยมีอินพุทพารามิเตอร์ได้หลายตัวแปร ก็ควรจะใช้วิธีการเลือกสัญญาณอินพุทมากกว่าการจัดสร้างหลายชุด) ตามรูป 4.11 เป็น NEXT_INDEX สำหรับตัวแปร HEAD และ NEXT_INDEX สำหรับตัวแปร TAIL



รูปที่ 4.10 ใช้บัฟเฟอร์เลือกใช้ NEXT_INDEX



รูปที่ 4.11 ใช้ NEXT_INDEX แยกจากกัน

เช่นเดียวกันกับ NEXT_INDEX สำหรับวงจรที่ใช้เปรียบเทียบค่าที่ใช้กับ Q_EMPTY และ Q_FULL ในหัวข้อ 1.3 ก็อาจจัดสร้างใน 2 ลักษณะเช่นกัน กล่าวคืออาจสร้างวงจรเปรียบเทียบชุดเดียวแล้วสร้างวงจรสำหรับเลือกสัญญาณอินพุทหรือสร้างเป็น 2 ชุดสำหรับ Q_EMPTY และ Q_FULL แยกต่างหากจากกัน แต่อาจมีข้อแม้ว่าผลจาก Q_EMPTY และ Q_FULL นั้นจะต้องไม่ใช้พร้อมกันหรือใช้ตลอดเวลา

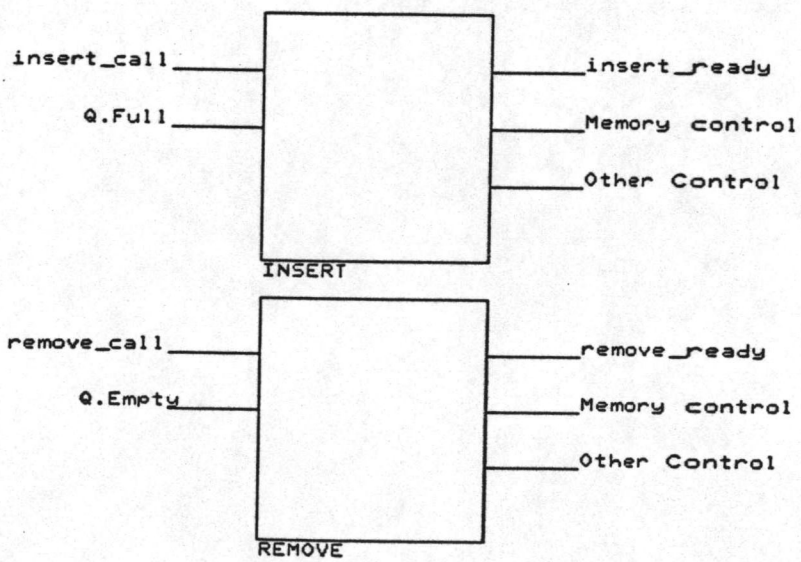
1.5 <u>อัลกอริธึม Q_INSERT และ Q_REMOVE</u> อัลกอริธึม Q_INSERT และ Q_REMOVE เป็นอัลกอริธึมที่มีลำดับของการทำงานเป็นขั้นตอนหลายลำดับที่ต้องเกี่ยวข้องกับโครงสร้างข้อมูลและอัลกอริธึมที่ได้กล่าวมาแล้วในหัวข้อ 1.1-1.4 ซึ่งในการออกแบบจะเริ่มต้นที่ผังภาพกล่องดำของอัลกอริธึมเหล่านี้ที่จะแสดงการติดต่อกับภายนอก และความสัมพันธ์กับอัลกอริธึมและโครงสร้างข้อมูลอื่น ๆ

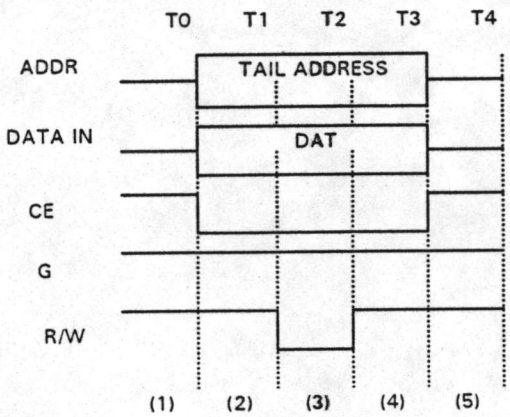
เราจะพิจารณาอัลกอริธึม Q_INSERT ทีละคำสั่งโดยเริ่มต้นจากคำสั่ง

Q.QUEUE [Q.TAIL] := DAT ;

คำสั่งนี้เป็นการนำข้อมูลจาก DAT เข้าไปเก็บในอาเรย์ ณ ตำแหน่งที่ตัวแปร TAIL ชี้อยู่ซึ่งจะต้องพิจารณากระบวนการบันทึกข้อมูลลงในหน่วยความจำจากผังเวลาในภาพที่ 4.2 (ข) และ (ค) เราจะใช้ผังเวลาสำหรับการบันทึกข้อมูลในแบบ (ข) คือแบบที่สัญญาณ OUTPUT ENABLE (G) ไม่ได้อยู่ในสภาวะ 0 (Active) ตลอดเวลา ซึ่งผังเวลาที่ได้ออกแบบสำหรับการบันทึกข้อมูลแสดงดังภาพที่ 4.13

รูปที่ 4.12 แผนผังกล่องดำของอัลกอริธึม Q_INSERT และ Q_REMOVE





รูปที่ 4.13 ผังเวลาควบคุมการทำงานของหน่วยความจำ

RAM ให้บันทึกข้อมูลจาก DAT ลงในตำแหน่งที่อ้างถึงโดย TAIL

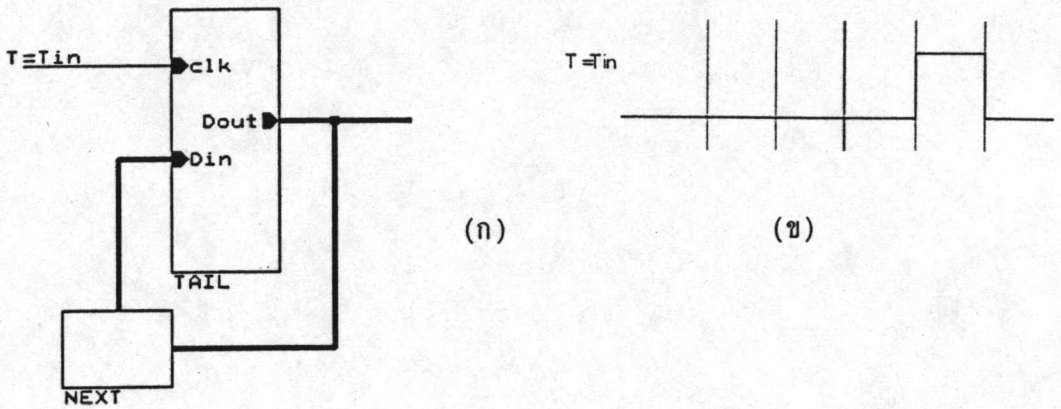
1. ให้สัญญาณ TAIL เข้าไปยัง ADDRESS ของ RAM ส่งสัญญาณ DAT เข้าไปยัง DATA ของ RAM. .ให้สัญญาณ CE เพื่อให้ RAM เริ่มรับคำสั่งซึ่งภายใน RAM จะเลือกตำแหน่ง ADDRESS จาก TAIL

2. ให้สัญญาณ WRITE (R/W = 0) เพื่อให้ RAM นำข้อมูลจาก DAT เข้าไปบันทึกในตำแหน่งที่อ้างด้วย TAIL

3.ยกเลิกสัญญาณ WRITE (R/W = 1) เนื่องจาก G เป็น 1 ตลอดเวลาข้อมูลจาก DAT เข้าไปบันทึกในตำแหน่งที่อ้างด้วย TAIL

4. เมื่อ RAM ออกจากสภาวะการบันทึกโดยสมบูรณ์แล้วจะยกเลิกสัญญาณ ADDRESS จาก TAIL, DATA จาก DAT และ CE เพื่อให้หน่วยความจำเข้าสู่สภาวะ STANDBY

คำสั่งถัดไปได้แก่คำสั่ง Q.TAIL := NEXT_INDEX (Q.TAIL); ทำได้โดยการให้สัญญาณนาฬิกาไปยัง REGISTER ที่เป็นตัวแปร TAIL เพื่อให้รับข้อมูลเข้าที่เราต่อไว้กับ NEXT_INDEX (Q. TAIL) ดังรูปที่ 4.14 ซึ่งจะต้องกระทำหลังจากคำสั่งแรกเสร็จสิ้นแล้วเมื่อพิจารณาผังเวลาจากภาพ 4.13 แล้วจะพบว่าในช่วงเวลาที่ 3 ถึงช่วงเวลาที่ 4 เป็นช่วงที่การบันทึกข้อมูลจาก DAT ลงในหน่วยความจำเสร็จสิ้นแล้ว ในจังหวะดังกล่าวไม่มีการใช้ตัวแปร TAIL เพื่ออ้างตำแหน่งจึงอาจให้สัญญาณนาฬิกาแก่ตัวแปร TAIL ในจังหวะเวลาที่ 3 ถึง 4 ได้ ซึ่งเราจะต้องต่อวงจรของตัวแปร TAIL กับ NEXT_INDEX (TAIL) ดังรูปที่ 4.14 (ก) และผังเวลาในรูป 4.14 (ข)



รูปที่ 4.14 ส่วนวงจร Q.TAIL := NEXT_INDEX (Q.TAIL) (ก) การต่อวงจรเพื่อรับข้อมูล (ข) ผังเวลา

จากที่กล่าวมาจะพบว่าลำดับของการทำงานในอัลกอริธึม Q_INSERY แบ่งได้เป็น 4 ลำดับ ซึ่งในแต่ละลำดับจะมีความสัมพันธ์กับการสร้างสัญญาณต่าง ๆ ดังนี้

$$TAIL_{index} = (S_1 - S_3)_{Now\_Insert}$$

$$DAT_{enable} = (S_1 - S_3)_{Now\_Insert}$$

$$CE = (S_1 - S_3)_{Now\_Insert}$$

$$G = 1$$

$$R/W = (S_2)_{Now\_Insert}$$

$$T = Tin = S_4{}_{Now\_Insert}$$

โดย

ตัวห้อย $_{Now\ Insert}$ แสดงว่าอยู่ระหว่างอัลกอริธึมการเพิ่มข้อมูล

S1-S3 หมายถึงช่วงระหว่างลำดับเวลาที่ 1 ถึงลำดับที่ 3 จะเป็นลอจิก 1

S_1-S_3 จะได้ลอจิกตรงข้ามกับ S1 - S_3

S_2 หมายถึง จะให้ลอจิก 0 ขณะที่อยู่ในช่วงเวลาลำดับที่ 2

S_4 หมายถึง จะให้ลอจิก 1 ขณะที่อยู่ในช่วงเวลาลำดับที่ 4

$TAIL_{index}$ คือสัญญาณที่จะควบคุมให้บัฟเฟอร์ของ TAIL ทำงาน

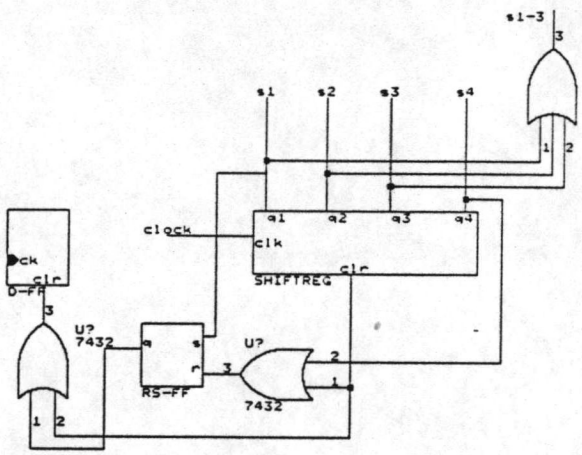
$DATA_{enable}$ คือสัญญาณควบคุมให้บัฟเฟอร์ข้อมูลเข้าทำงาน

CE, G, R/W คือ สัญญาณควบคุมของหน่วยความจำ

$T = T_{in}$ คือสัญญาณนาฬิกาที่ควบคุมให้ตัวแปร TAIL รับข้อมูลจากอินพุท

การสร้างลำดับการทำงานของ Q_INSERT จะต้องได้รับสัญญาณจากภายนอก INSERT_CALL เรียกใช้อัลกอริธึมนี้จึงจะเริ่มต้นทำงาน และในระหว่างที่ยังทำงานอยู่นั้นจะให้สัญญาณ NOW_INSERT ที่แสดงว่ากำลังทำงานในอัลกอริธึม Q_INSERT นี้อยู่ สัญญาณการเรียกใช้จะถูกตรวจจับที่ขอบขาขึ้นของสัญญาณ ดังนั้นสัญญาณที่เรียกใช้อัลกอริธึมนี้จะต้องเป็นพัลซ์การเรียกใช้ครั้งต่อไปจะต้องส่งสัญญาณพัลซ์มาเรียกใช้หลังจากการทำงานครั้งก่อนเสร็จสิ้นแล้ว

การสร้างสัญญาณที่เป็นลำดับอาจใช้วงจรชิพรีจิสเตอร์เป็นตัวสร้างลำดับ หรือใช้วงจรนับและถอดรหัสเป็นตัวสร้างลำดับก็ได้ หากใช้ชิพรีจิสเตอร์สร้างลำดับ 4 ลำดับอาจทำได้ดังภาพ 4.15



รูปที่ 4.15 การใช้วงจรชิพรีจิสเตอร์เพื่อสร้างลำดับการทำงาน 4 ลำดับ

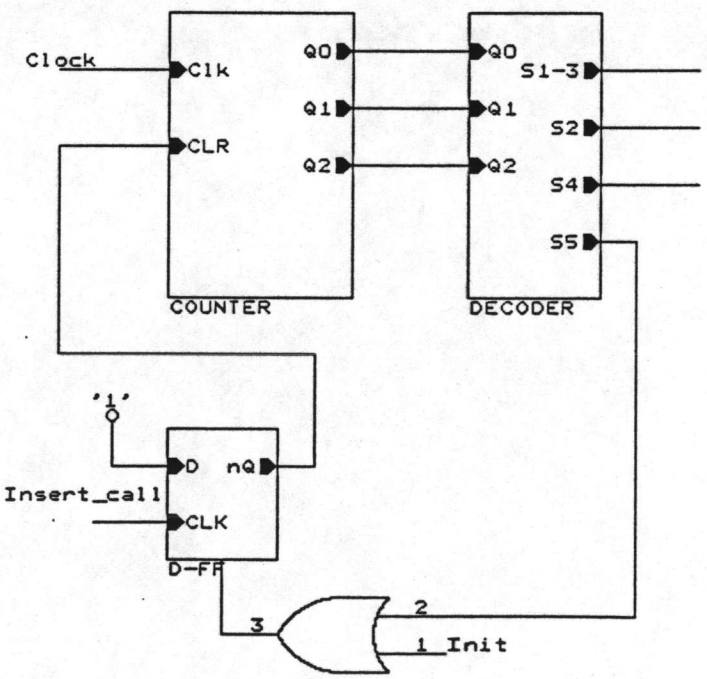
จากภาพ 4.15 สภาวะเริ่มต้น (โดยสัญญาณ INIT) จะทำให้ S_1-S_4 เป็น 0 ทั้งหมด รวมทั้งตัวแปร NOW_INSERT ก็จะถูกเคลียร์เป็น 0 ด้วย นอกจากนี้ยังได้นำสัญญาณ INIT ไปทำการเคลียร์สัญญาณ INSERT_CALL ที่ผ่านอุปกรณ์ INS-FF ด้วย เพื่อไม่ให้เกิดการเรียกใช้อัลกอริธึมนี้โดยไม่เจตนาภายหลังสภาวะเริ่มต้น

เมื่อได้รับสัญญาณ INSERT_CALL เป็นกระตุ้นให้ INS-FF รับข้อมูล 1 จากอินพุท D ไปปรากฏที่เอาต์พุท Q กลุ่มชิพรีจิสเตอร์จะได้รับข้อมูล 1 เริ่มการขยับข้อมูล 1 นี้ เข้าไปตามจังหวะสัญญาณนาฬิกาเมื่อได้รับข้อมูล 1 เลื่อนเข้าไปที่ S1 ก็จะเซทตัวแปร NOW_INSERT ซึ่งใช้ RS-FlipFlop ให้เป็น 1 จะทำการเคลียร์ INS-FF ให้สัญญาณเอาต์พุท Q เป็น 0 ค้างไว้ ซึ่งสัญญาณนาฬิกาจังหวะถัดไปจะเลื่อนข้อมูล 0 เข้าไป

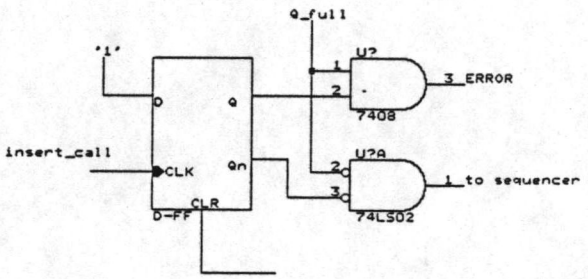
ในรีจิสเตอร์จะทำให้มี 1 เพียงหลักเดียวเลื่อนอยู่ในกลุ่มของชิพรีจิสเตอร์จนกระทั่งข้อมูล 1 นี้ถูกเลื่อนออกจาก S_4 เข้าไปในรีจิสเตอร์ถัดไปจะทำการรีเซทตัวแปร NOW_INSERT ให้เป็น 0 ยกเลิกการเคลียร์ INS-FF ให้พร้อมรับ INSERT_CALL ต่อไป

หากจะใช้วงจรนับและถอดรหัสเพื่อสร้างสัญญาณลำดับก็อาจทำได้โดยใช้วงจรนับเลขฐาน 2 (Binary counter) และวงจรถอดรหัสให้ได้สัญญาณที่สามารถนำไปใช้โดยง่ายคือ (S_1-S_3). S_3และ S_4 ซึ่งสามารถจัดทำได้ดังรูป 4.16 ซึ่งจะต้องใช้วงจรนับเลขฐานสองขนาด 3 บิต (นับได้ตั้งแต่ 0-7) แต่เราจะออกแบบให้มีการนับเพียง 0-4 เมื่อนับต่อเป็น 5 จะทำการเคลียร์ตัวนับให้เป็น 0 เริ่มต้นการทำงานในครั้งแรกสุดวงจรจะได้รับสัญญาณ INIT ซึ่งจะมีผลให้เอาต์พุท Q ของ CALL-FF ถูกเคลียร์เป็น 0 ซึ่งจะทำให้วงจรนับถูกเคลียร์เป็น 000 ด้วย และจะค้างในสภาวะนี้แม้ว่าสัญญาณ Init จะไม่ถูกส่งมาแล้ว จนกระทั่งได้รับสัญญาณเป็นพัลซ์จาก INS_CALL จะกระตุ้นให้เอาต์พุท Q ของ CALL_FF เป็น 1 วงจรนับก็จะเริ่มทำงานตามจังหวะสัญญาณนาฬิกา (CLOCK) นับตั้งแต่ 001 จนกระทั่งถึง 101 ก็จะทำให้สัญญาณ S5 เป็น 1 ซึ่งจะทำให้ CALL-FF ถูกเคลียร์ มีผลให้วงจรนับถูกเคลียร์เป็น 000 ค้างไว้ด้วยจนกระทั่งได้รับสัญญาณ INSERT_CALL ใหม่อีกครั้ง ในระหว่างการนับ 001-100 และเกิดการเคลียร์ที่ 101 นั้น สัญญาณ NOW_INSERT จะเป็น 1 (NOW_INSERT = S_0; $S_0 = Q_0 \Lambda Q_1 \Lambda Q_2$)

รูปที่ 4.16 การใช้วงจรนับเลขฐาน 2 และถอดรหัสเป็นวงจรสร้างสัญญาณลำดับ

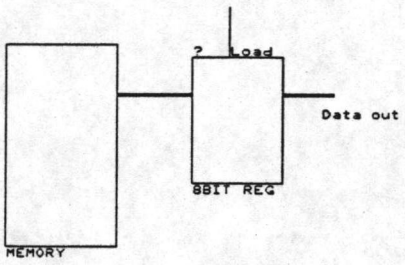


การตรวจสอบคำสั่ง IF NOT Q_FULL นั้น สามารถทำได้โดยจะกำหนดเพิ่มเติมหากมีการเรียกใช้ Q_INSERT ในขณะที่ข้อมูลในแถวคอยเต็มอยู่จะให้เกิดสัญญาณ ERROR ขึ้นซึ่งอาจถือว่าเป็นตัวแปรตัวหนึ่งก็ได้ ดังนั้นหากเกิดสัญญาณ ERROR ขึ้นจะต้องไม่มีการทำงานตามลำดับที่กล่าวมาแล้วข้างต้นด้วย เราจะดัดแปลงวงจรที่รับสัญญาณ INSERT_CALL ใหม่ ดังรูป 4.17 โดยถือว่า ERROR = CALLED Λ Q_FULL

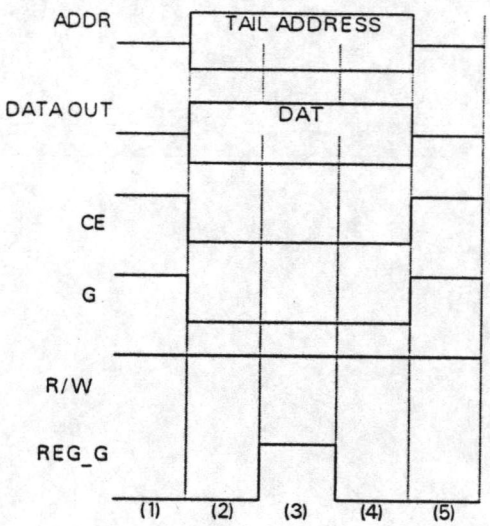


รูปที่ 4.17 วงจรตรวจสอบ IF NOT Q_FULL และ ERROR

การพิจารณาอัลกอริธึม Q_REMOVE ก็เช่นเดียวกับ Q_INSERT โดยเริ่มจากคำสั่ง DAT := Q.QUEUE [Q.HEAD] เป็นคำสั่งอ่านข้อมูลจากหน่วยความจำของอาเรย์ ณ ตำแหน่งที่ตัวแปร HEAD ชี้อยู่ซึ่งข้อมูลที่อ่านออกมาได้เราจะใช้รีจิสเตอร์เก็บข้อมูลนั้นไว้ดังรูป 4.18 ในการเก็บข้อมูลโดยรีจิสเตอร์ก็จะต้องมีการสร้างสัญญาณ INPUT ENABLE (G) ให้แก่รีจิสเตอร์เพื่อรับข้อมูลมาเก็บไว้ ในการออกแบบวงจรเราจะอาศัยผังเวลาตามภาพที่ 4.2 (ก) เป็นข้อมูลเบื้องต้นในการออกแบบซึ่งสามารถจะเขียนผังเวลาการทำงานในการอ่านข้อมูลจากหน่วยความจำเก็บลงในอุปกรณ์รีจิสเตอร์ได้ ดังรูปที่ 4.19



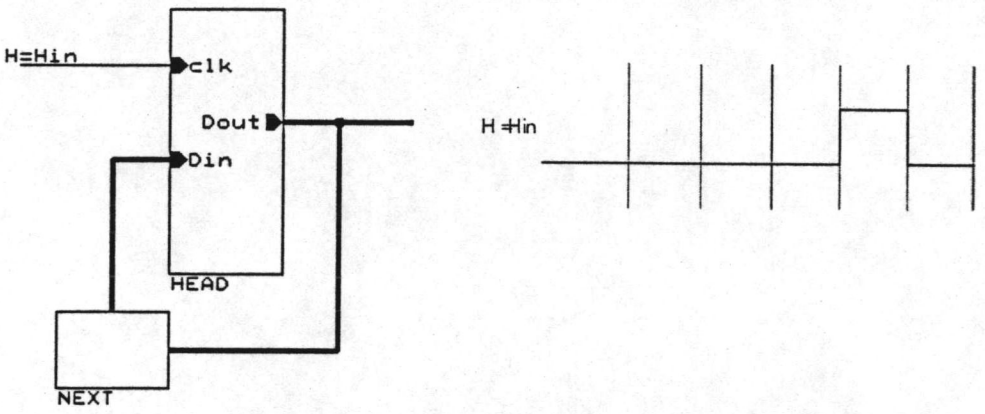
รูปที่ 4.18 การใช้รีจิสเตอร์เพื่อรับข้อมูลออกจากอาเรย์



รูปที่ 4.19 ผังเวลาของการนำข้อมูลออกจากอาเรย์

คำสั่งถัดมาได้แก่คำสั่ง Q.HEAD := NEXT_INDEX (Q.HEAD) สามารถทำได้ในทำนองเดียวกับการเพิ่มค่า TAIL ในภาพที่ 4.14 คือการให้สัญญาณแก่ CLOCK ของตัวแปร HEAD เพื่อให้ตัวแปร HEAD รับเข้าข้อมูล HEAD +1 ในช่วงเวลา S_4 หลังจากที่เลิกการใช้หน่วยความจำแล้ว ภาพผังเวลาของสัญญาณ $H = H_{in}$ และภาพการต่อวงจรแสดงดังรูปที่ 4.20

รูปที่ 4.20 คำสั่ง Q.HEAD := NEXT_INDEX(Q.HEAD) (ก) การต่อวงจร (ข) ผังเวลา



(ก) (ข)

การสร้างสัญญาณต่าง ๆ เพื่อควบคุมการนำข้อมูลออกจากแถวคอยที่สัมพันธ์กับลำดับเวลาต่าง ๆ สามารถเขียนได้ดังนี้

$HEAD_{index}$ = S1-S3 Now_Remove

CE = S1-S3 Now_Remove

G = S1-S3 Now_Remove

R/W = 1_{Now_Remove}

REG_G = $S2_{Now\_Remove}$

$H=H_{in}$ = $S4_{Now\_Remove}$

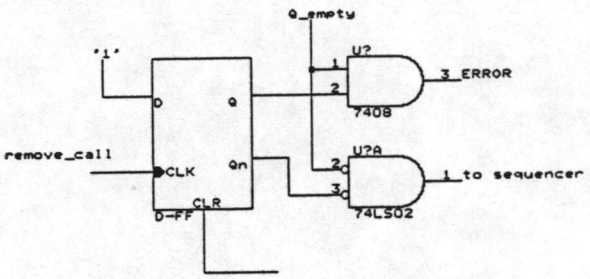
โดย

$HEAD_{index}$ หมายถึงสัญญาณควบคุมบัฟเฟอร์ของ HEAD

$H=H_{in}$ หมายถึงสัญญาณนาฬิกาควบคุมให้ตัวแปร HEAD รับข้อมูล

REG_G หมายถึงสัญญาณ INPUT ENABLE ของรีจิสเตอร์ซึ่งใช้เก็บข้อมูลที่ได้รับจากการ REMOVE

จากภาพ 4.19 และ 4.20 จะเห็นว่าจำนวนลำดับเวลาของการดึงข้อมูลออกจะมี 4 จังหวะเวลาเช่นเดียวกับการเพิ่มข้อมูล ดังนั้นจึงสามารถใช้วงจรสร้างลำดับเช่นเดียวกับ Q_INSERT ได้ ส่วนคำสั่ง IF NOT Q_EMPTY ก็สามารถออกแบบในลักษณะเดียวกับ IF NOT Q_FULL ได้ ดังภาพที่ 4.21



รูปที่ 4.21 วงจร IF NOT Q_EMPTY

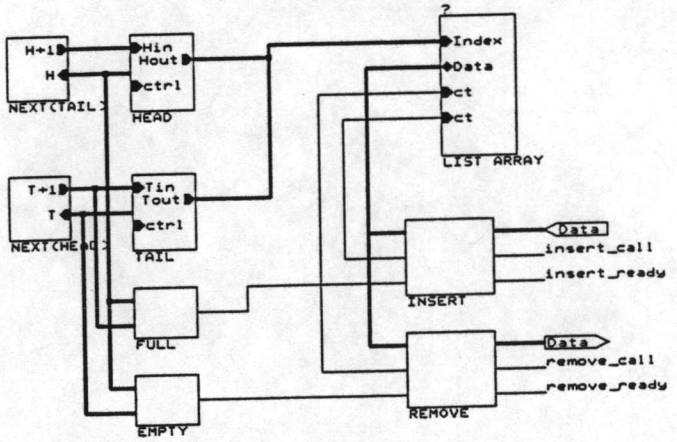
เนื่องจากทั้งอัลกอริธึม Q_REMOVE มีการให้สัญญาณเพื่อควบคุมหน่วยความจำดังนั้น จึงต้องพิจารณาการให้สัญญาณควบคุมแก่หน่วยความจำจากทั้ง 2 อัลกอริธึม ซึ่งสามารถแสดงได้ดังนี้

$$CE = (S1\text{-}S3)_{Now\_Insert} \vee (S1\text{-}S3)_{Now\_Remove}$$

$$G = (S1\text{-}S3)_{Now\_Remove}$$

$$R/W = S2_{Now\_Insert}$$

เมื่อเรารวมอัลกอริธึมทั้งสองเข้าด้วยกันสามารถแสดงผังวงจรได้ดังภาพที่ 4.22

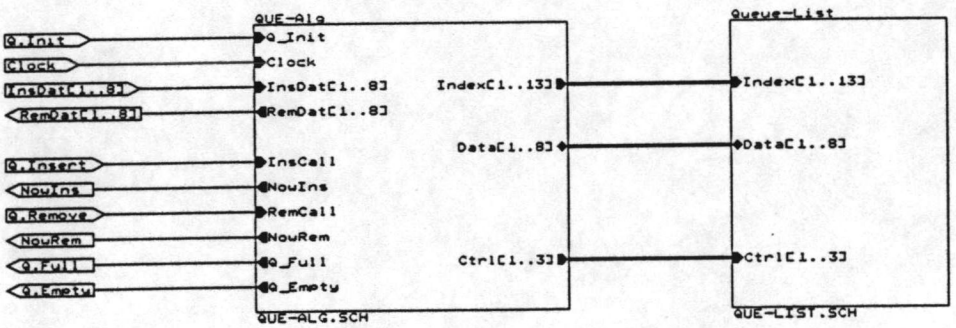


รูปที่ 4.22 วงจรสำหรับสัญญาณควบคุมอาเรย์

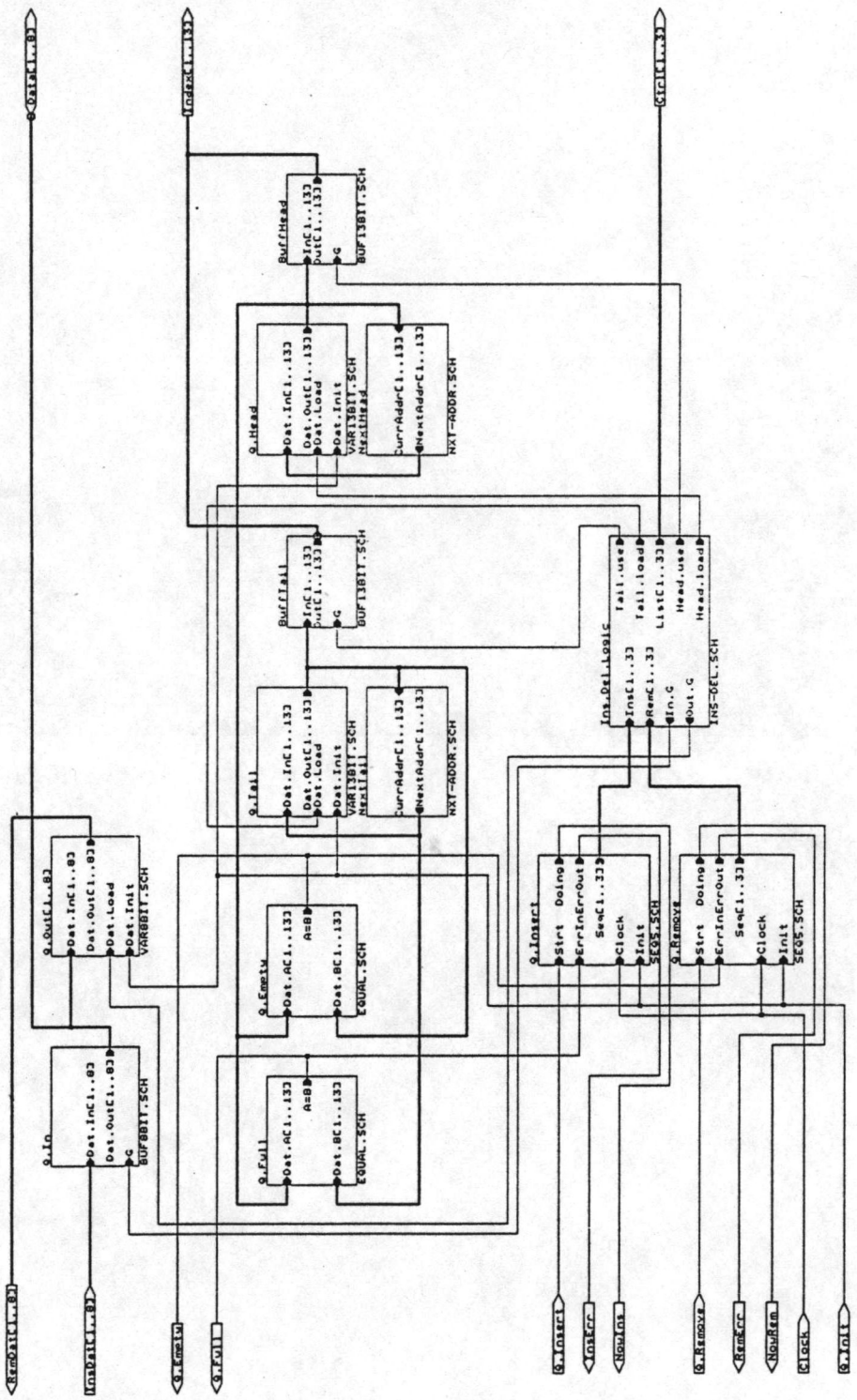
2 การจัดสร้างวงจรจัดการข้อมูลแถวคอยโดยใช้วงจรรวมตระกูล 74

การจัดสร้างวงจรจัดการแถวคอยที่ได้ออกแบบในหัวข้อ 4.1 โดยใช้วงจรรวมในตระกูล 74 สามารถทำได้โดยเลือกอุปกรณ์วงจรรวมที่เหมาะสม ซึ่งจะต้องมีการเปลี่ยนแปลงวงจรในบางส่วนเล็กน้อยเพื่อให้เข้ากับอุปกรณ์ที่มีอยู่สำหรับอุปกรณ์วงจรรวมในตระกูล 74 นี่จะเลือกใช้เป็นอุปกรณ์ในระดับที่มีความซับซ้อนต่ำและปานกลาง (SSI และ MSI) ในโครงการนี้ได้ทำการทดลองโดยใช้โปรแกรมออแคด (ORCAD) เพื่อป้อนวงจรเลือกใช้อุปกรณ์จากไลบรารีของวงจรรวมตระกูล 74LS และได้ทำการจำลองผลการทำงาน ซึ่งได้ผลดังหัวข้อที่จะกล่าวต่อไป

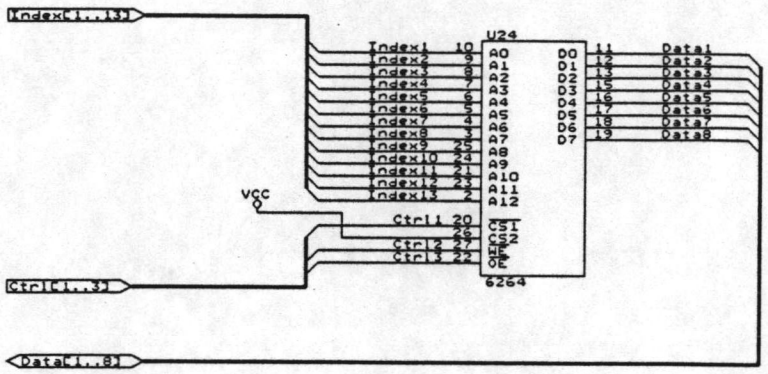
ในการทดลองจัดสร้างวงจรจัดการแถวคอยได้เลือกอุปกรณ์ต่าง ๆ และป้อนแบบวงจรโดยใช้โปรแกรมออแคดแสดงดังผังวงจรรูปที่ 4.23 ถึง 4.33 เมื่อใช้คำสั่ง Create Bill of Material จะได้อุปกรณ์ต่าง ๆ ดังรายการในรูปที่ 4.34



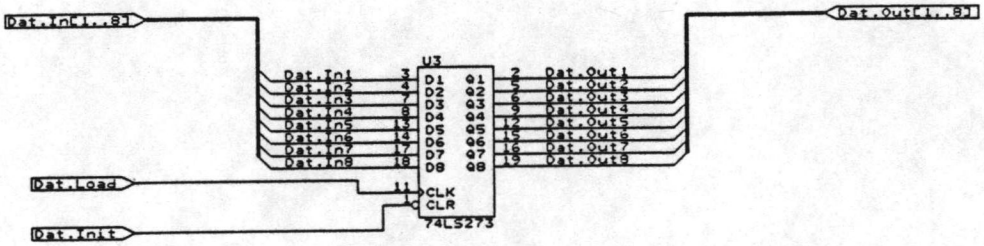
รูปที่ 4.23 แผนภาพวงจรหลักของวงจรจัดการข้อมูลแถวคอยซึ่งประกอบด้วยวงจรย่อย 2 ส่วนได้แก่ ส่วนวงจร Queue Managment และ List Array



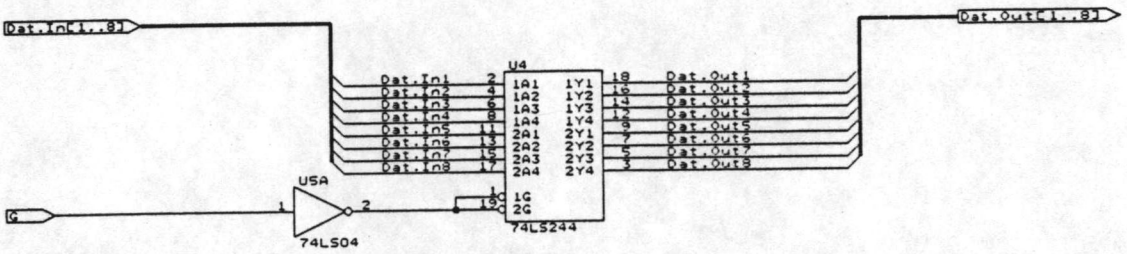
รูปที่ 4.24 แผนภาพวงจรย่อย Queue Management ซึ่งประกอบด้วยวงจรย่อยต่างๆ



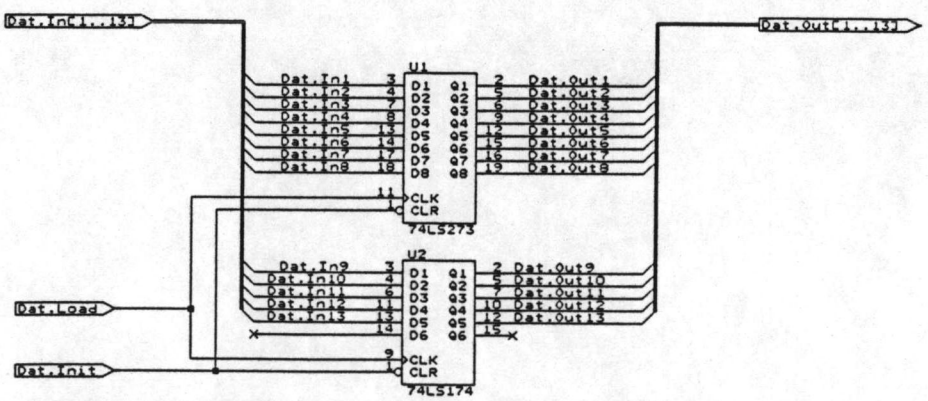
รูปที่ 4.25 แผนภาพวงจรย่อย List Array ซึ่งใช้หน่วยความจำ Static RAM



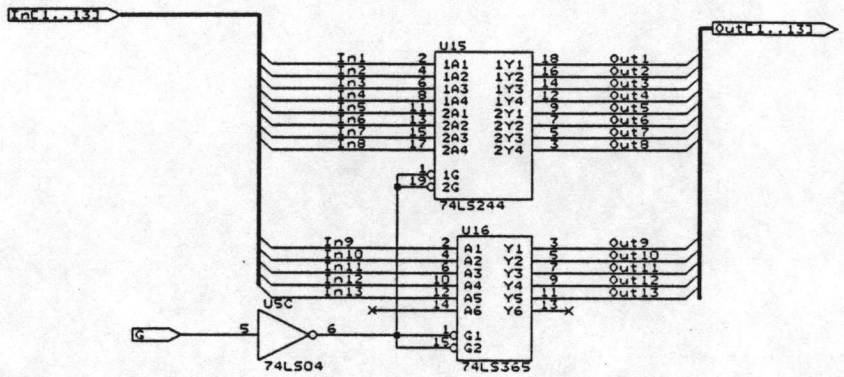
รูปที่ 4.26 ส่วนวงจรย่อย VAR8BIT ที่ใช้เป็นรีจิสเตอร์สำหรับเก็บข้อมูล DATA OUT



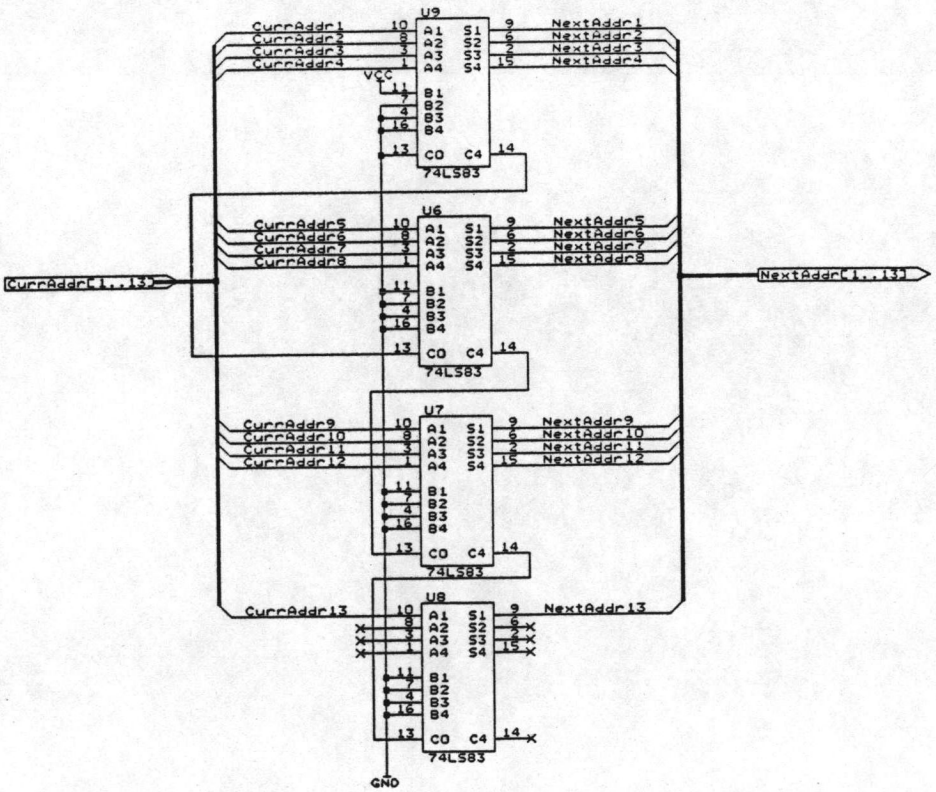
รูปที่ 4.27 ส่วนวงจรย่อย BUF8BIT ที่ใช้ต่อเชื่อมกับ DATA IN



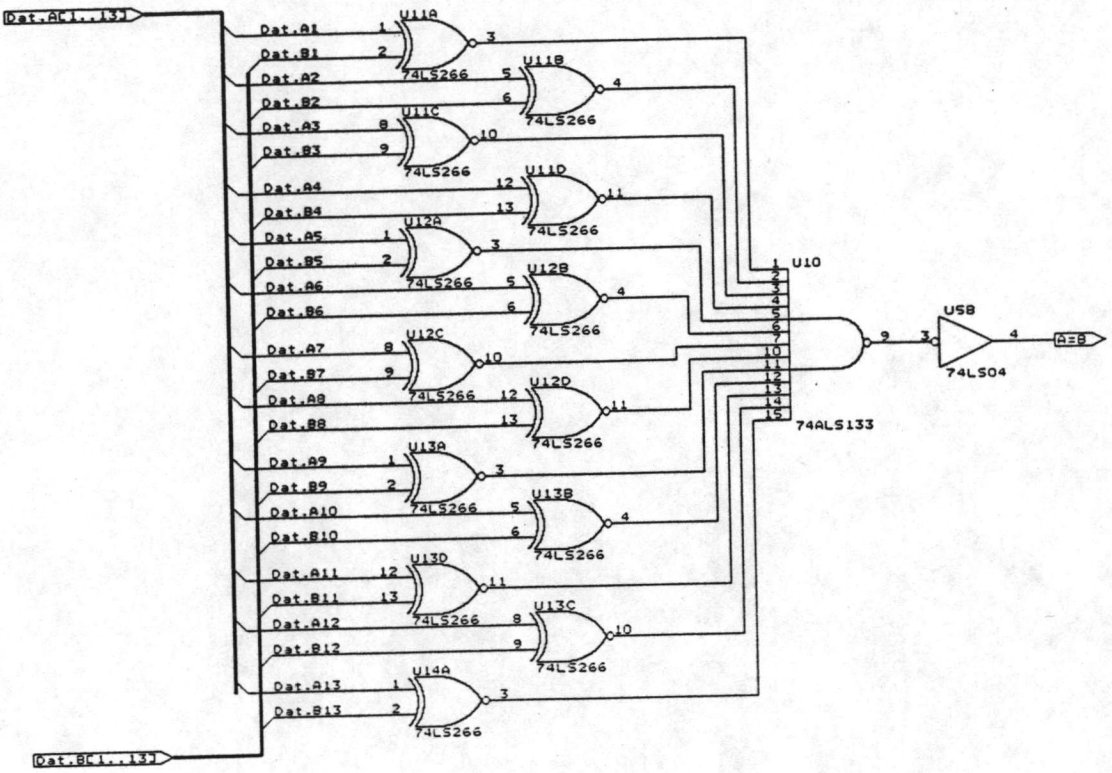
รูปที่ 4.28 ส่วนวงจรย่อย VAR13BIT ที่ใช้เป็นตัวแปร HEAD และ TAIL



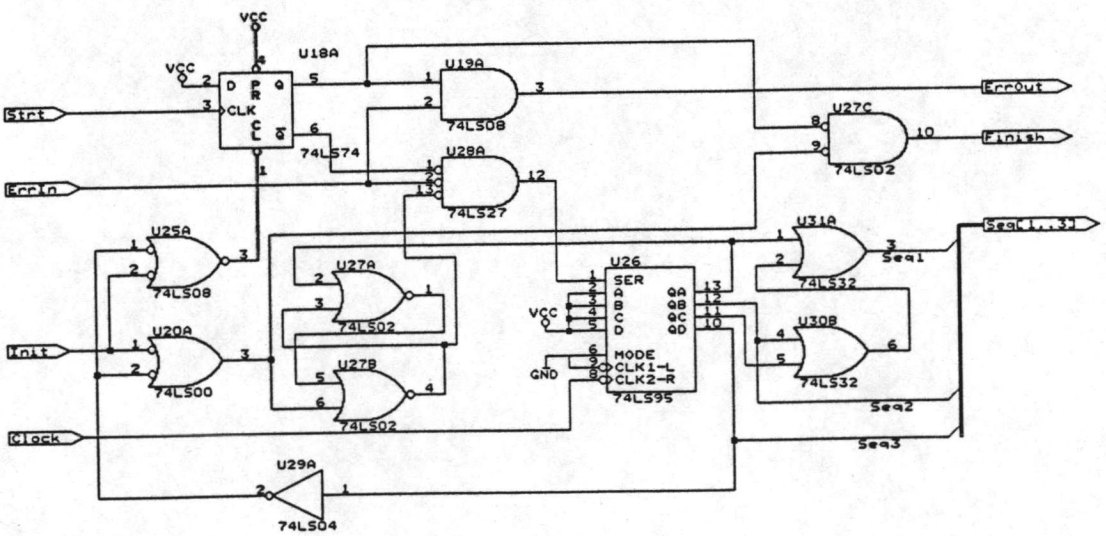
รูปที่ 4.29 ส่วนวงจรย่อย BUFF13BIT ใช้เลือกอินเดกว์จากตัวแปร HEAD และ TAIL



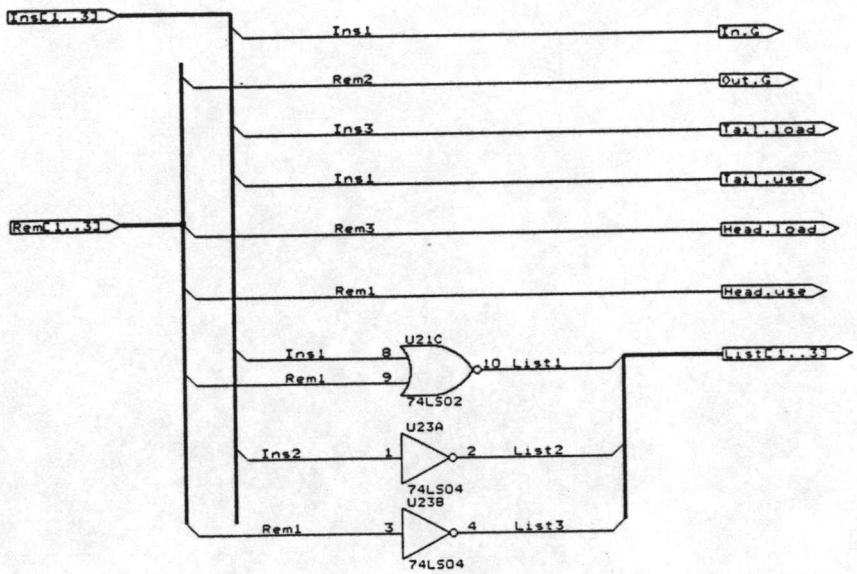
รูปที่ 4.30 ส่วนวงจรย่อย NEXT_INDEX ที่ใช้กับร่วมตัวแปร HEAD และ TAIL



รูปที่ 4.31 ส่วนวงจรย่อย EQUAL ที่ใช้เปรียบเทียบอินเดกซ์สำหรับ Q.EMPTY และ Q.FULL



รูปที่ 4.32 ส่วนวงจรย่อย SEQ5 เป็นส่วนที่ใช้สร้างสัญญาณลำดับการทำงานทั้ง REMOVE และ INSERT



รูปที่ 4.33 ส่วนวงจรย่อย INS_DEL เป็นส่วนวงจรสร้างสัญญาณสำหรับควบคุมอาเรย์และรีจิสเตอร์ต่างๆ

Queue DataStructure (root sheet) Revised: March 7, 1994
Revision:
Bill Of Materials March 19, 1994 15:00:46 Page 1

Item	Quantity	Reference	Part
1	2	U1,U3	74LS273
2	1	U2	74LS174
3	2	U4,U15	74LS244
4	3	U5,U23,U29	74LS04
5	4	U6,U7,U8,U9	74LS83
6	1	U10	74ALS133
7	4	U11,U12,U13,U14	74LS266
8	1	U16	74LS365
9	1	U18	74LS74
10	2	U19,U25	74LS08
11	1	U20	74LS00
12	2	U21,U27	74LS02
13	1	U24	6264
14	1	U26	74LS95
15	1	U28	74LS27
16	1	U30	74LS32

รูปที่ 4.34 รายการอุปกรณ์วงจรรวมมาตรฐานในตระกูล 74LSxx ที่ใช้กับวงจรจัดการข้อมูลแถวคอย

3. <u>การทดสอบผลการทำงานโดยใช้โปรแกรมจำลองผลวงจร</u>

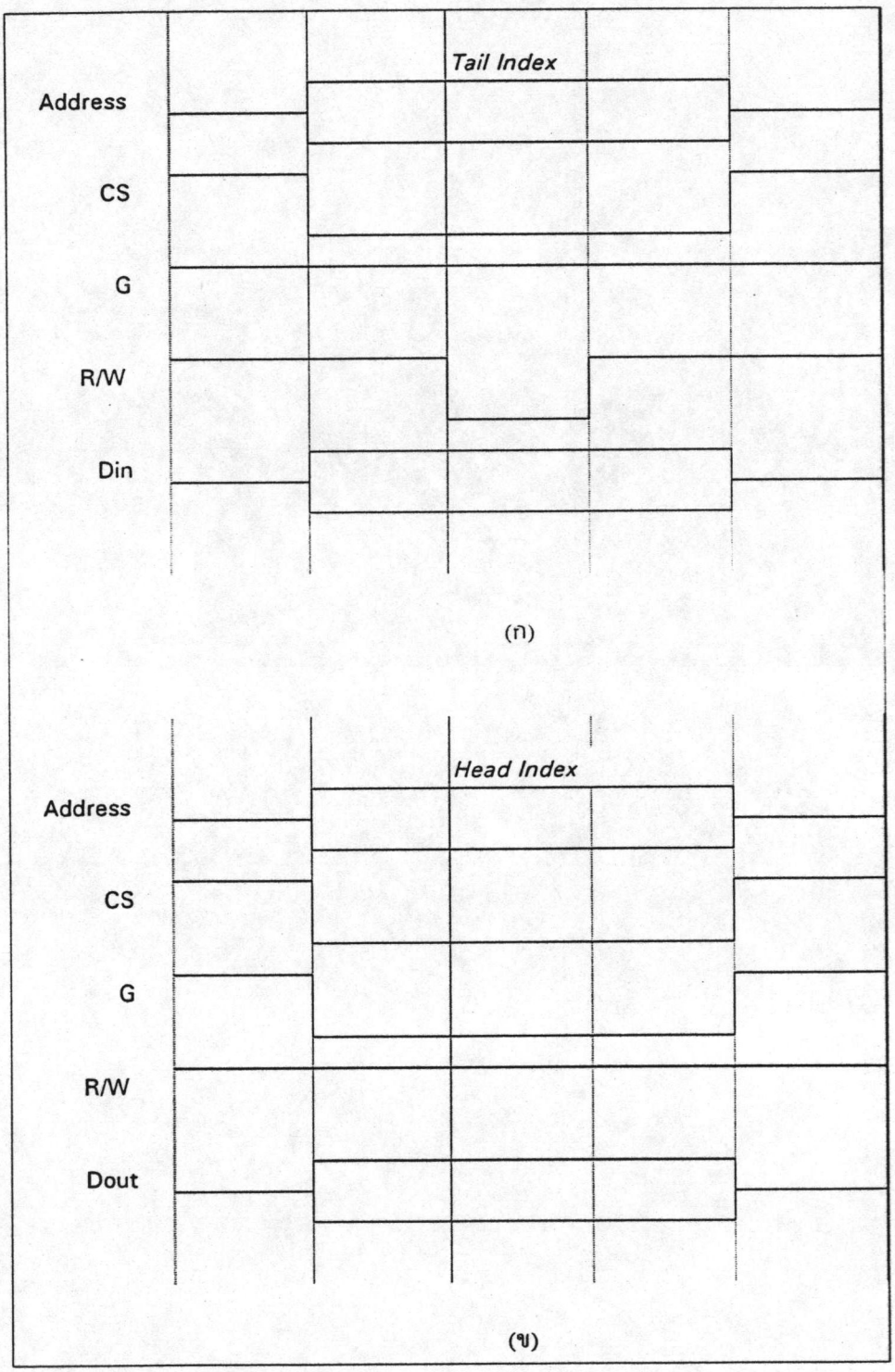
การทดสอบผลการทำงานของวงจรในโครงการนี้จะใช้โปรแกรมจำลองผลการทำงานซึ่งจะตั้งเงื่อนไขเพื่อจำลองผลการทำงานในสภาวะต่าง ๆ ได้แก่

1) การอ่าน-เขียน ข้อมูลกับหน่วยความจำ
2) สภาวะของการตั้งต้น (Initialigation State)
3) สภาวะของการเพิ่มข้อมูลหลังการตั้งต้น
4) สภาวะของการเพิ่มข้อมูลที่ยังไม่ทำให้ข้อมูลเต็มแถวคอย
5) สภาวะของการเพิ่มข้อมูลแล้วทำให้ข้อมูลเต็มแถวคอย
6) สภาวะของการเพิ่มข้อมูลเมื่อข้อมูลเต็มแถวคอยอยู่ก่อนแล้ว
7) สภาวะของการนำข้อมูลออกเมื่อมีข้อมูลอยู่ในแถวคอยมากกว่า 1 ตัว
8) สภาวะของการนำข้อมูลออกจากแถวคอยแล้วแถวคอยว่าง
9) สภาวะของการนำข้อมูลออกจากแถวคอยเมื่อแถวคอยว่างอยู่ก่อน

ผลการจำลองการทำงานวงจรจัดการข้อมูลแถวคอย ที่ได้จากโปรแกรมการจำลองการทำงานแสดงผลดังรูปที่ 4.36 ถึงภาพ 4.43

3.1 <u>การอ่าน-เขียนกับหน่วยความจำ</u> เนื่องจากในวงจรจัดการข้อมูลแถวคอยต้องใช้หน่วยความจำทำหน้าที่เป็นอาเรย์ ซึ่งในการจำลองผลจะจำลองผลเฉพาะส่วนของวงจรจัดการ โดยจะตรวจสอบผลการจำลองที่ได้จากวงจรจัดการแถวคอยให้ได้ตรงตามเงื่อนไขตามรูปคลื่นของการอ่าน-เขียนหน่วยความจำ ตามมาตรฐานที่ระบุในคู่มือของหน่วยความจำ ซึ่งจะนำภาพของผังเวลาการทำงานอ่าน-เขียนหน่วยความจำที่ได้ศึกษาจากรูปที่ 4.2 ในหัวข้อ 1.1 ของบทที่ 4 นี้มาใช้เป็นรูปสัญญาณสำหรับการควบคุมหน่วยความจำที่ใช้เป็นอาเรย์

การกำหนดรูปสัญญาณการควบคุมหน่วยความจำในการอ่าน-เขียนแสดงในรูปที่ 4.35 โดยรูปที่ 4.35 (ก) เป็นรูปสัญญาณการบันทึกข้อมูลลงหน่วยความจำซึ่งจะใช้กับสถานะการเพิ่มข้อมูลลงในแถวคอย (queue insert) ส่วนรูปที่ 4.35 (ข) เป็นรูปสัญญาณการอ่านข้อมูลจากหน่วยความจำซึ่งจะใช้กับสถานะการนำข้อมูลออกจากแถวคอย (queue remove)

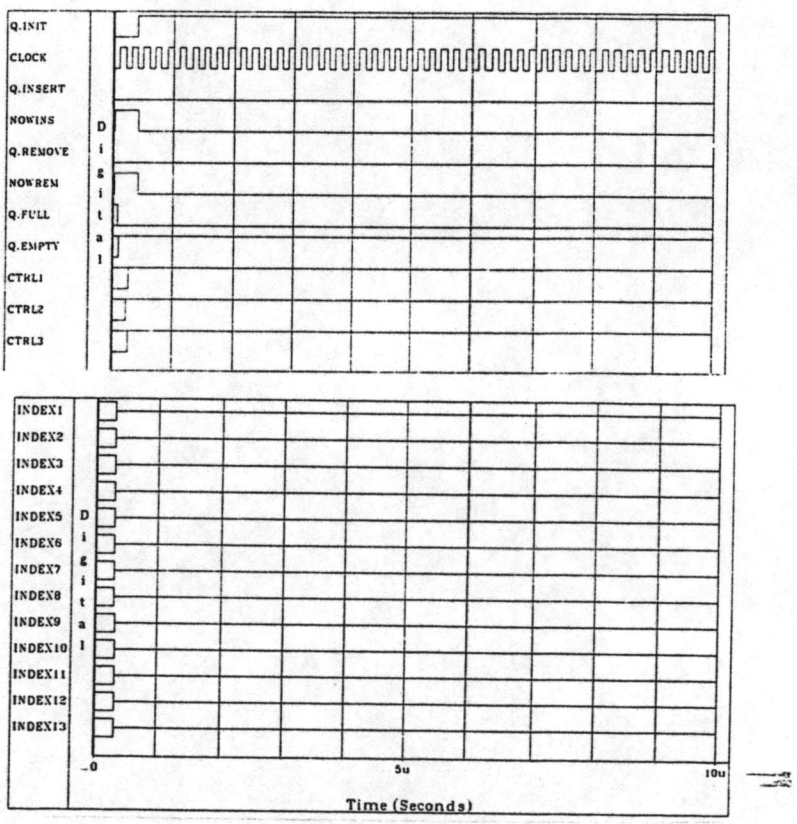


รูปที่ 4.35 รูปสัญญาณที่ใช้ควบคุมหน่วยความจำ (ก) การบันทึกข้อมูล (ข) การอ่านข้อมูล

3.2 <u>การจำลองผลในสภาวะตั้งต้น</u> สภาวะตั้งต้น (initialisation state) คือ สภาวะที่ป้อนสัญญาณ RESET (RESET = 0) ให้แก่วงจรจัดการข้อมูลแถวคอย จนกระทั่งยกเลิกการให้สัญญาณ RESET ซึ่งเปรียบเทียบได้กับการเรียก Q_INITIAL ในหัวข้อ 3.4 ผลของการเรียกใช้ Q_INTIAL จะทำให้ตัวแปร HEAD และ TAIL มีค่าข้อมูลเป็น 0 (ตำแหน่งตั้งต้น) นอกเหนือจากนี้ในวงจรผังต้องทำการปรับอุปกรณ์ส่วนต่าง ๆ ให้เข้าสู่สภาวะเริ่มต้นด้วย เช่น วงจรสร้างสัญญาณลำดับ และวงจรอื่น ๆ ที่ใช้อุปกรณ์ฟลิบฟลอบ ฯลฯ

ในการกำหนดสัญญาณเข้าให้แก่วงจรจะกำหนดสัญญาณนาฬิกา (clock) ที่มีความถี่คงที่ที่ 4 เมกกะเฮริทซ์ ซึ่งการนำไปใช้งานจริงนั้นความถี่ของสัญญาณนาฬิกาจะขึ้นกับองค์ประกอบอื่น ๆ เฉพาะงานนั้น ๆ และความถี่สูงสุดจะต้องไม่เกินกว่าความสามารถของอุปกรณ์ที่จะใช้ว่าสามารถทำงานได้ (ทั้งวงจรจัดการและหน่วยความจำ) การกำหนดสัญญาณ RESET ให้ทำการตั้งต้นระบบ RESET=0 เป็นระยะเวลา 1 รอบสัญญาณนาฬิกา (จากวงจรที่ได้ออกแบบสัญญาณ RESET อาจมีช่วงระยะเวลาที่สั้นได้ไม่น้อยกว่า 25 นาโนวินาที ซึ่งเป็นระยะเวลาของอุปกรณ์ที่ทีแอลในตระกูล 74LS)

ผลของการจำลองการทำงานในสภาวะตั้งต้นนี้แสดงในรูปที่ 4.36

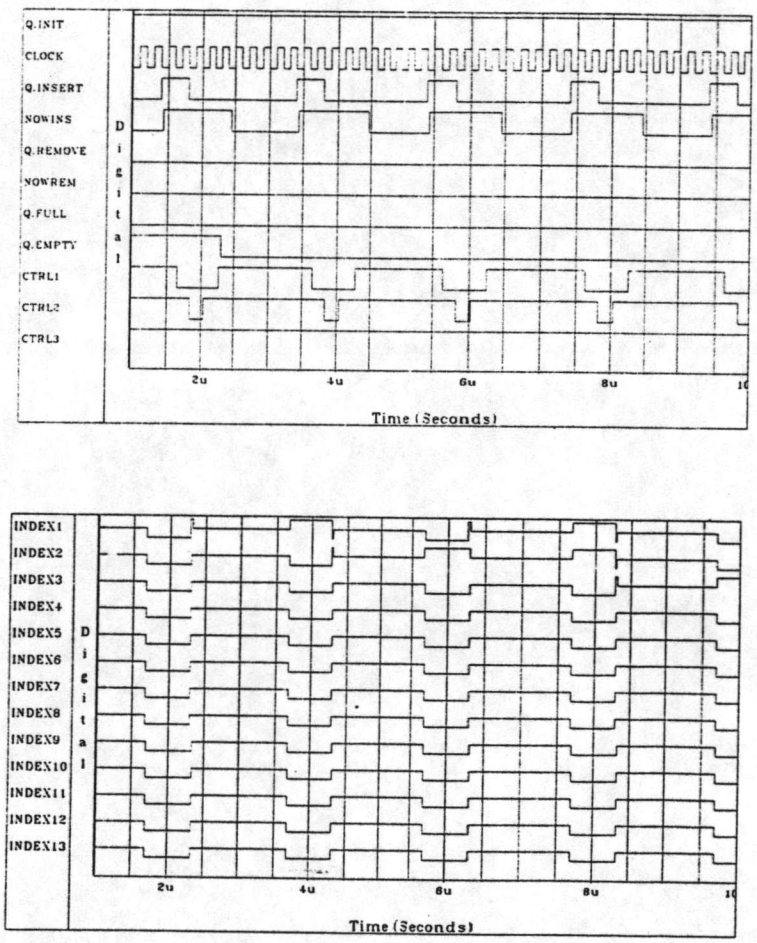


รูปที่ 4.36 ผลของการจำลองการทำงานในสภาวะตั้งต้น

3.3 <u>การจำลองผลการเพิ่มข้อมูลหลังการตั้งต้น</u> การกำหนดสภาวะของการเพิ่มข้อมูลหลังการตั้งต้นเป็นการให้สัญญาณเพิ่มเติมข้อมูล (INSERT_CALL) เป็นสัญญาณพัลซ์เมื่อ HEAD และ TAIL เป็น 0 สัญญาณ RESET ไม่แอคทีฟ และดูผลลัพธ์จนกระทั่งจบกระบวนการเพิ่มข้อมูล

ผลของการจำลองการทำงานแสดงดังรูป 4.37

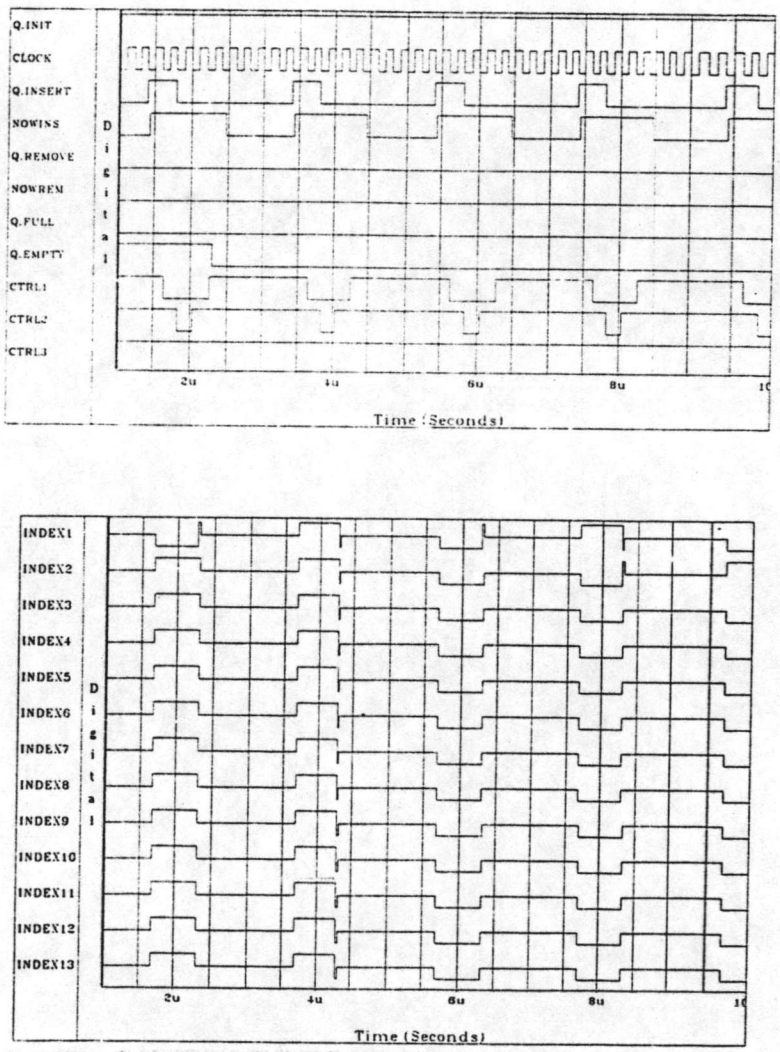
รูปที่ 4.37 ผลของการจำลองการทำงานการเพิ่มข้อมูลหลังสภาวะตั้งต้น



รูปที่ 4.37 ผลของการจำลองการทำงานการเพิ่มข้อมูลหลังสภาวะตั้งต้น

3.4 <u>การจำลองผลการเพิ่มข้อมูลที่ยังไม่ทำให้ข้อมูลเต็มแถวคอย</u> การกำหนดสภาวะการเพิ่มข้อมูลที่ยังไม่ทำให้ข้อมูลเต็มแถวคอย โดยกำหนดสภาวะเริ่มต้นของการจำลองผลให้ตัวแปร HEAD มีค่า 1 FFE_{16} และ TAIL มีค่า $1FFE_{16}$ และทำการเพิ่มข้อมูล 3 ครั้ง ซึ่งจะทำให้เกิดสภาวะ EMPTY, TAIL > HEAD และ TAIL < HEAD ที่ยังไม่ทำให้แถวคอยเต็ม

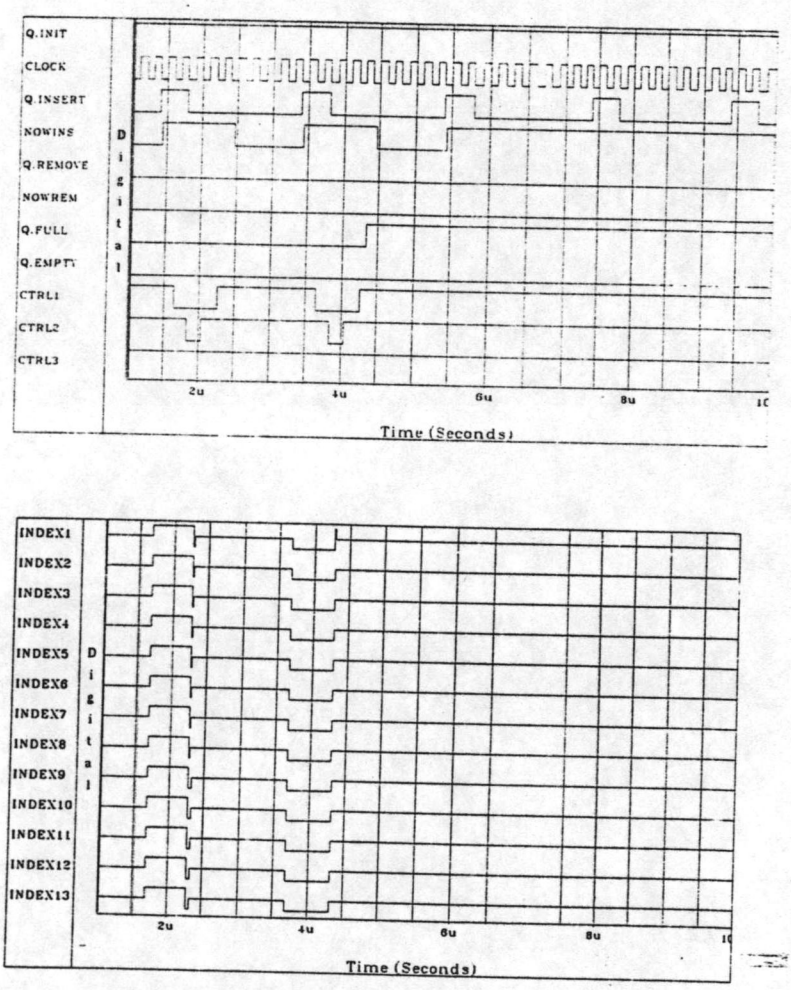
ผลของการจำลองแสดงได้ดังรูป 4.38



รูปที่ 4.38 ผลการจำลองการเพิ่มข้อมูลที่ยังไม่ทำให้แถวคอยเต็ม

3.5 <u>การจำลองผลสภาวะของการเพิ่มข้อมูลแล้วทำให้แถวคอยเต็ม</u> การกำหนดเงื่อนไขในสภาวะของการเพิ่มข้อมูลแล้วทำให้แถวคอยเต็ม โดยกำหนดให้สภาวะเริ่มต้นของการจำลองผลให้ตัวแปร HEAD มีค่า 0002_{16} และTAIL มีค่า $1FFF_{16}$ ทำการเพิ่มข้อมูล 2 ครั้ง ซึ่งจะทำให้เกิดสภาวะ TAIL > HEAD, TAIL < HEAD, FULL

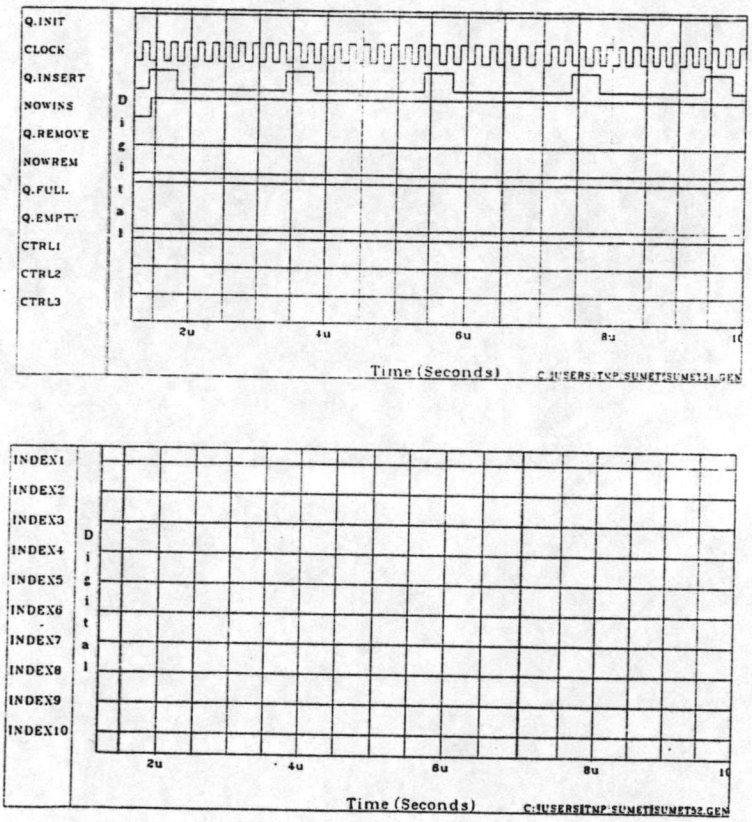
ผลของการจำลองแสดงดังรูป 4.39



รูปที่ 4.39 ผลการจำลองการเพิ่มข้อมูลแล้วทำให้แถวคอยเต็ม

3.6 <u>การจำลองผลในสภาวะของการเพิ่มข้อมูลเมื่อแถวคอยเต็มแล้ว</u> การกำหนดเงื่อนไขในสภาวะการเพิ่มข้อมูลเมื่อแถวคอยเต็มแล้วจะกำหนดให้ตัวแปร TAIL มีค่าเริ่มต้น 0001_{16} และ HEAD มีค่าเริ่มต้น 0002_{16} และทำการเพิ่มข้อมูล 2 ครั้ง

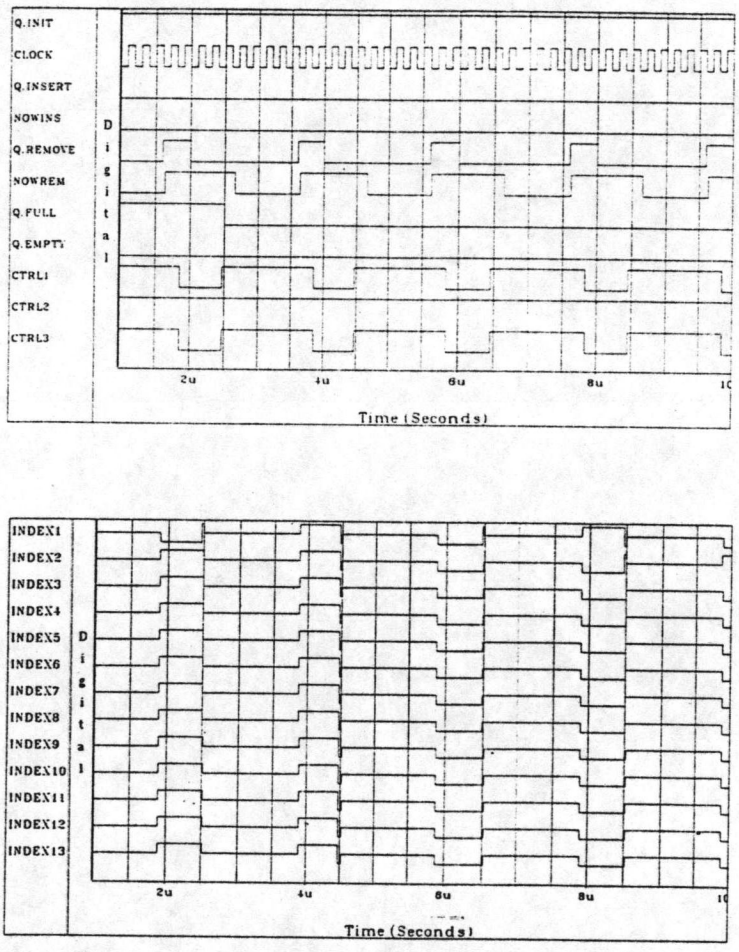
ผลของการจำลองแสดงดังรูป 4.40



รูปที่ 4.40 ผลการจำลองการเพิ่มข้อมูลเมื่อแถวคอยเต็มแล้ว

3.7 <u>การจำลองผลการนำข้อมูลออก เมื่อมีข้อมูลในแถวคอยมากกว่า 1 ตัว</u> การกำหนดสภาวะการนำข้อมูลออกเมื่อมีข้อมูลในแถวคอยมากกว่า 1 ตัว โดยกำหนดสภาวะเริ่มต้นให้ตัวแปร HEAD มีข้อมูล $1FFF_{16}$ และ TAIL มีข้อมูล $1FFD_{16}$ และทำการดึงข้อมูลออก 3 ครั้ง ซึ่งจะทำให้เกิดสภาวะ FULL (ขณะเริ่มต้น), HEAD > TAIL และ HEAD < TAIL โดยยังมีข้อมูลในแถวคอย

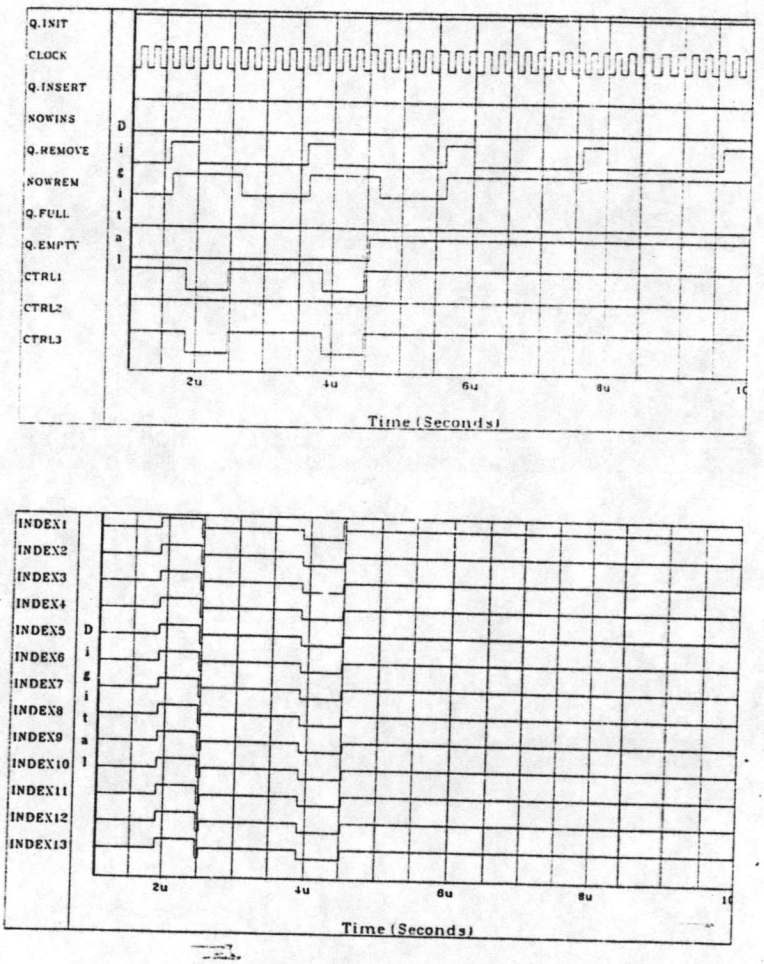
ผลของการจำลองแสดงดังรูป 4.41



รูปที่ 4.41 ผลการจำลองการนำข้อมูลออกเมื่อมีข้อมูลในแถวคอยมากกว่า 1 ตัว

3.8 <u>การจำลองผลการนำข้อมูลออกแล้วทำให้แถวคอยว่าง</u> กำหนดเงื่อนไขการนำข้อมูลออกแล้วทำให้แถวคอยว่างโดยกำหนดสภาวะเริ่มต้นให้ตัวแปร HEAD มีค่า $1FFF_{16}$ และ TAIL มีค่า 0001_{16} แล้วดึงข้อมูลออก 2 ครั้ง จะทำให้เกิดสภาวะ HEAD > TAIL (ขณะเริ่มต้น), HEAD < TAIL และ EMPTY

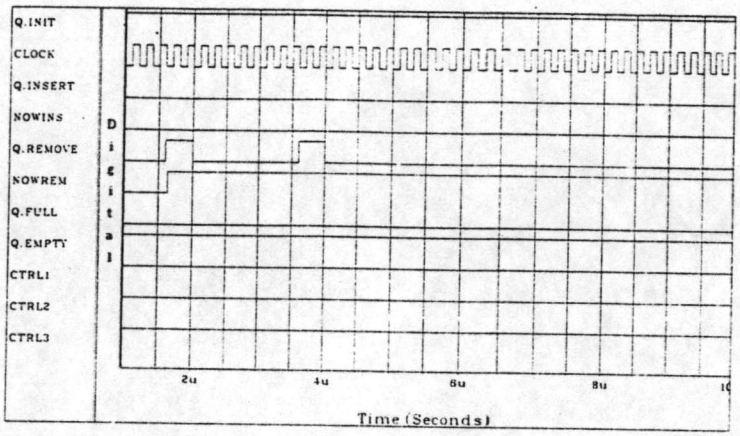
ผลของการจำลองแสดงดังรูป 4.42

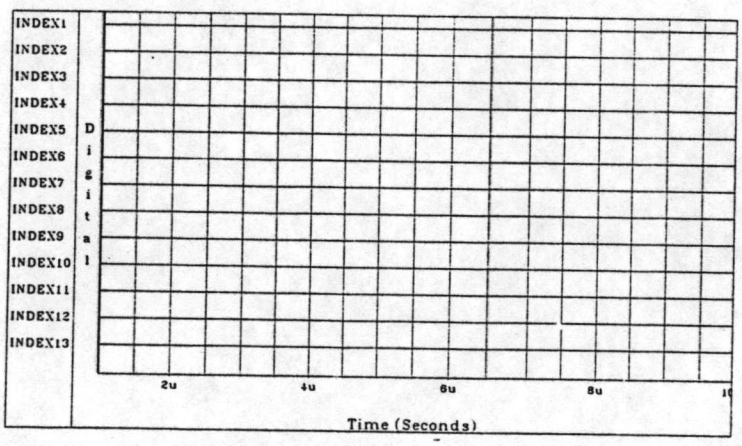


รูปที่ 4.42 ผลการจำลองการนำข้อมูลออกแล้วทำให้แถวคอยว่างเปล่า

3.9 <u>การจำลองผลการนำข้อมูลออกเมื่อแถวคอยว่างอยู่</u> การกำหนดเงื่อนไข การนำข้อมูลออกเมื่อแถวคอยว่างอยู่จะกำหนดสภาวะเริ่มต้นให้ HEAD และ TAIL มีค่าเริ่มต้น 0001_{16} และทำการ REMOVE ข้อมูล 2 ครั้ง

ผลของการจำลองแสดงดังรูป 4.43





รูปที่ 4.43 ผลการจำลองการนำข้อมูลออกจากแถวคอยเมื่อแถวคอยว่างอยู่

<u>สรุปผลการออกแบบวงจรจัดการแถวคอย</u>

จากที่ได้กล่าวมาในบทที่ 4 นี้ได้ทำการออกแบบวงจรจัดการข้อมูลแถวคอย และจำลองผลการทำงานของวงจรโดยใช้โปรแกรมออแคดใน 9 สภาวะ ได้ผลการจำลองตรงกับข้อกำหนดในการออกแบบ ซึ่งแสดงได้ว่าการออกแบบได้ผลตามความต้องการ

ในบทถัดไปเป็นการนำวงจรไปออกแบบวงจรรวมโดยใช้อุปกรณ์เกทอาเรย์ที่สามารถโปรแกรมได้ และวงจรรวมระดับหน้ากากทั้งฟูลคัสตอมและเซมิคัสตอมซึ่งใช้เซลมาตรฐาน