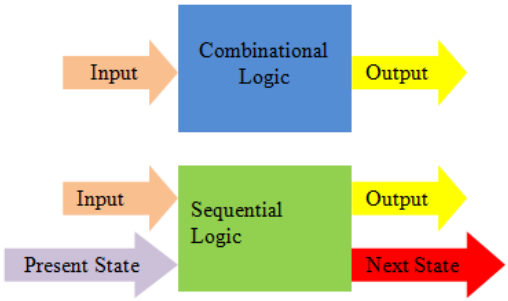
บทที่ 21 โครงสร้างการเขียนโปรแกรม

ทาง Digital แบ่ง Logic ตามหลักการทำงานได้เป็น 2 ชนิด คือ Combinational logic และ Sequential logic โดยที่ Combinational logic นั้นค่า output จะขึ้นอยู่กับค่าของ Input ปัจจุบันเพียงอย่างเดียวเท่านั้น ตรงกันข้ามกับ Sequential logic ที่ค่า output ไม่ได้ขึ้นอยู่กับค่าของ Input เพียงอย่างเดียวแต่ขึ้นกับสถานะ (State) ของตัวเองด้วย ดังรูปด้านล่าง



ในการเขียนโปรแกรม Ladder เราสามารถเขียนได้ทั้งแบบ Combination logic และแบบ Sequential logic โดยมีหลักการเหมือนกันคือแบ่งการทำงานของเครื่องจักรหรือกระบวนการออกเป็นส่วนย่อยๆตามฟังก์ชั่นการทำงาน ส่วนย่อยๆเหล่านั้นอาจจะทำงานเป็นอิสระหรืออินเตอร์ล็อคซึ่งกันและกันตามตัวอย่างด้านล่าง

ตัวอย่าง

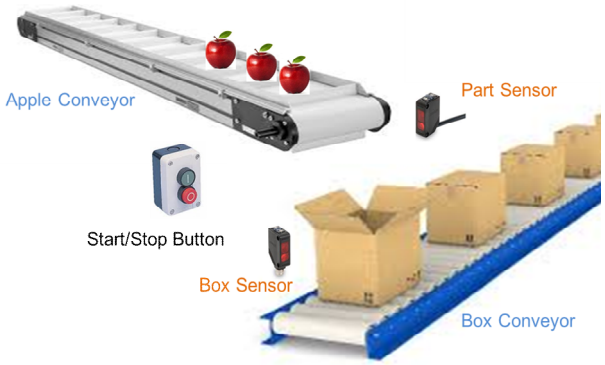
เครื่องใส่ผลแอ็ปเปิ้ลลงกล่อง ประกอบด้วย 2 สายพาน คือสายพานลำเลียงผลแอ็ปเปิ้ลและสายพานลำเลียงกล่อง โดยมีเซ็นเซอร์ SE1 นับจำนวนผลผลแอ็ปเปิ้ลและเซ็นเซอร์ SE2 เช็คตำแหน่งของกล่อง

เมื่อเครื่องอยู่ในโหมด Auto และพนักงานกดปุ่ม Start เครื่องจะเริ่มทำงาน โดยป้อนผลแอ็ปเปิ้ลลงกล่องกล่องละ 10 ผล เมื่อครบตามจำนวนแล้ว สายพานลำเลียงผลแอ็ปเปิ้ล จะหยุดรอจนกว่ากล่องเปล่าเลื่อนเข้ามายังตำแหน่งของ เซ็นเซอร์ SE2

จากนั้นสายพานลำเลียงผลแอปเปิ้ลจึงเริ่มทำงานอีกครั้ง เป็นอย่างนี้เรื่อยไป จนกว่าพนักงานกดปุ่ม Stop หรือมี Alarm เกิดขึ้น ได้แก่ ผลแอ็ปเปิ้ลติดขวางปากกล่อง, ไม่มีผลแอ็ปเปิ้ลลงกล่อง หรือไม่มีกล่องป้อนเข้ามาบนสายพาน

ถ้ามี Alarm เกิดขึ้น พนักงานต้องกดปุ่ม Reset เพื่อรีเซ็ต Alarm จากนั้นให้กดปุ่ม Start เครื่องจึงกลับมาทำงานใหม่อีกครั้ง

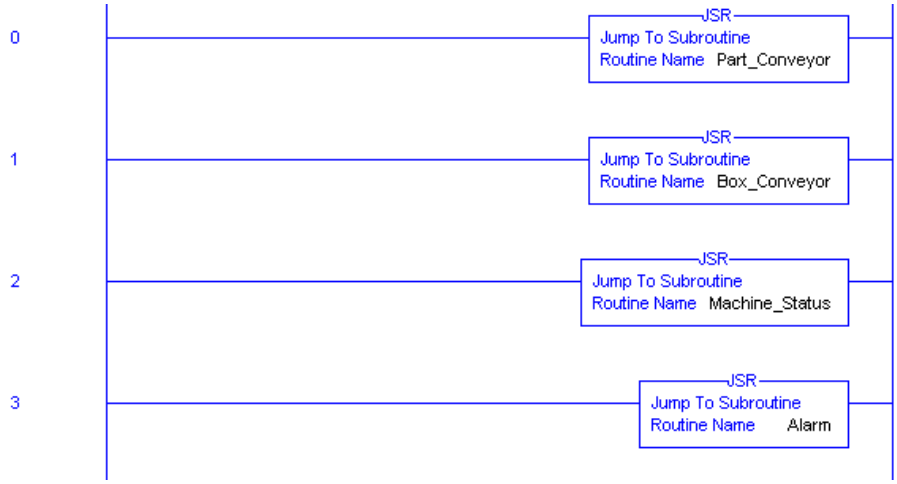
ในโหมด Manual พนักงานสามารถสั่งงานอุปกรณ์ได้โดยตรงจากการปุ่มกดควบคุมแต่ละอุปกรณ์



โปรแกรมแบบ Combination logic เอาท์พุตของอุปกรณ์จะทำงานตามอินพุตซึ่งเราจะแบ่งโปรแกรมออกเป็นส่วนย่อยๆ โดยใช้ Subroutine ดังนี้

- MainRoutine เป็นรูทีนหลักใช้ Jump ไป Subroutine ต่างๆ
- Alarm เป็นรูทีนสำหรับเขียนเงื่อนไข Alarm ของเครื่อง
- Box_Conveyor ใช้ควบคุมการทำงานของ Box conveyor
- Machine_Status ใช้แสดงสถานะของเครื่อง ได้แก่ Ready, Running, Fault, Auto และ Manual
- Part_Conveyor ใช้ควบคุมการทำงานของ Apple conveyor

MainRoutine



Alarm Routine

Rung 0 -> Alarm เนื่องจากผลแอ็บเปิ้ลติดขวางทางลงกล่องนานเกิน 5 วินาที

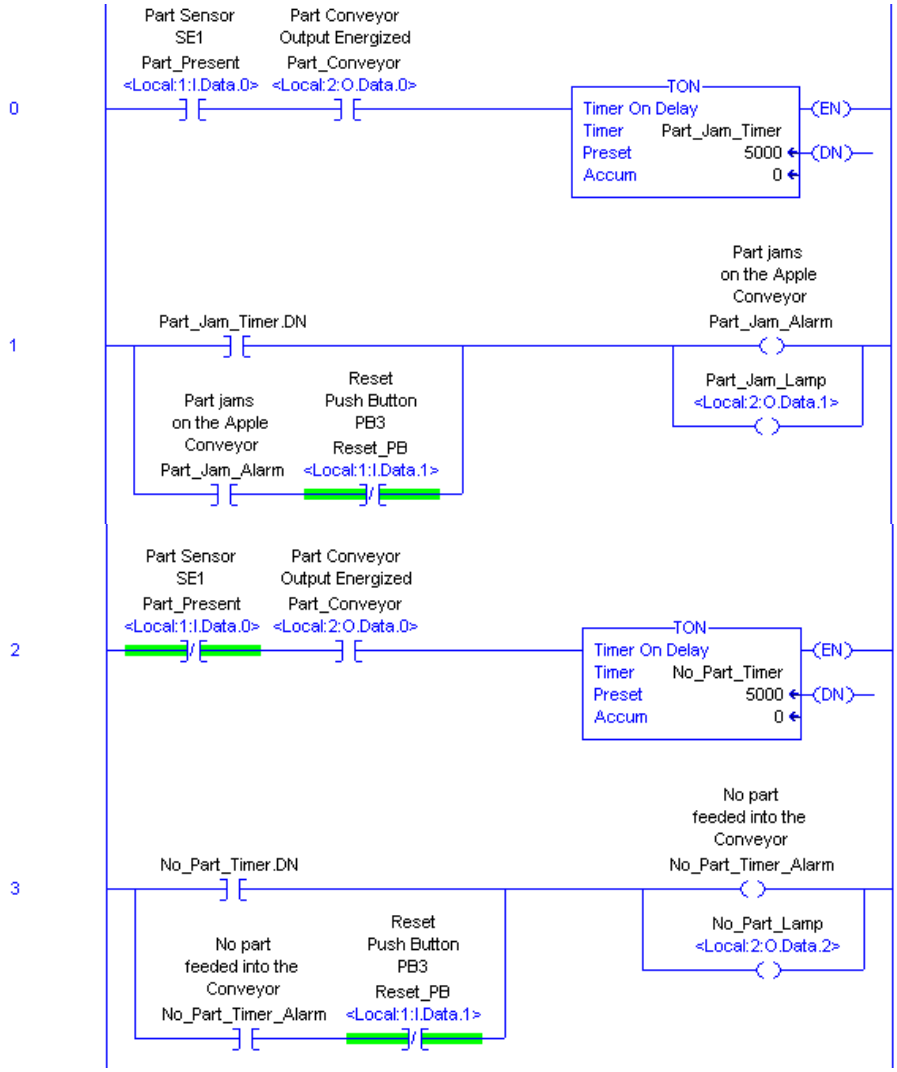
Rung 1 -> รีเซ็ต Alarm และเปิดหลอดไฟแสดง Alarm

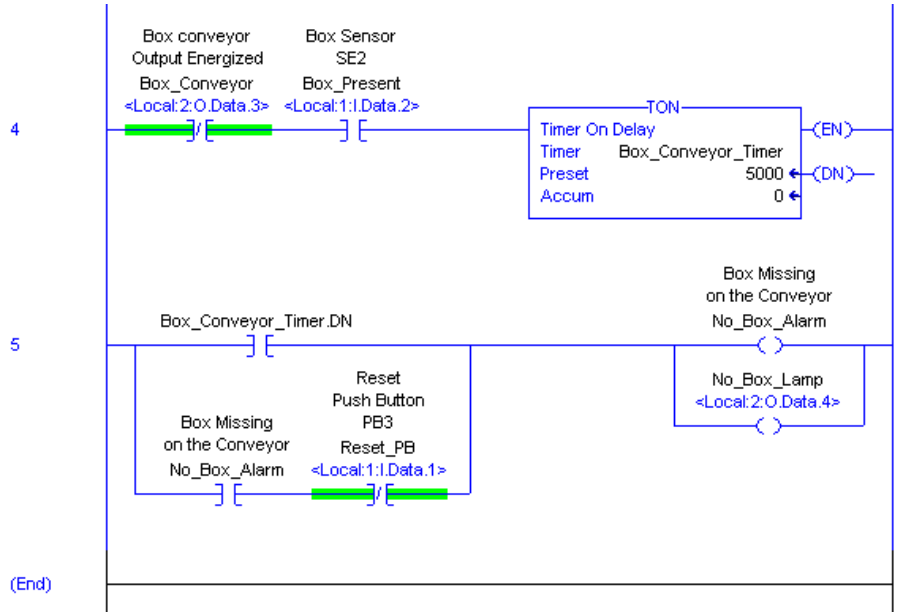
Rung 2 -> Alarm เนื่องจากไม่มีผลแอ็บเปิ้ลลงกล่องนานเกิน 20 วินาที

Rung 3 -> รีเซ็ต Alarm และเปิดหลอดไฟแสดง Alarm

Rung4 -> Alarm เนื่องจากไม่มีกล่องถูกป้อนเข้ามานานเกิน 2 วินาที

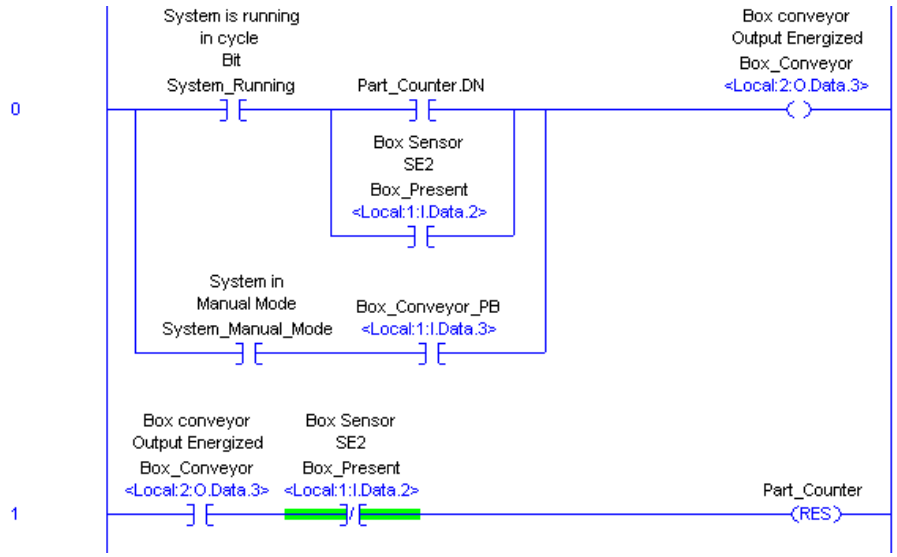
Rung 5 -> รีเซ็ต Alarm และเปิดหลอดไฟเตือน Alarm





Box_Conveyor

โปรแกรมแบบ Combination logic เอาท์พุตของอุปกรณ์ทำงานตามอินพุตดังนั้น Box_Conveyor จะทำงานเมื่อ Part_Counter นับผลแอ็ปเปิ้ลครบแล้ว หรือยังไม่มีกล่องอยู่ในตำแหน่ง (Box_Present มีสถานะเป็น Fault) อย่างไรก็ตามถ้าเครื่องหยุดรัน สายพานต้องหยุดทำงานด้วย เราจึงต้องนำบิต System_Running มาต่ออนุกรมเข้าไป ส่วน Part_Counter จะถูกรีเซ็ตหลังจากกล่องเคลื่อนที่พ้นเซ็นเซอร์ Box_Present แล้ว



Machine_Status

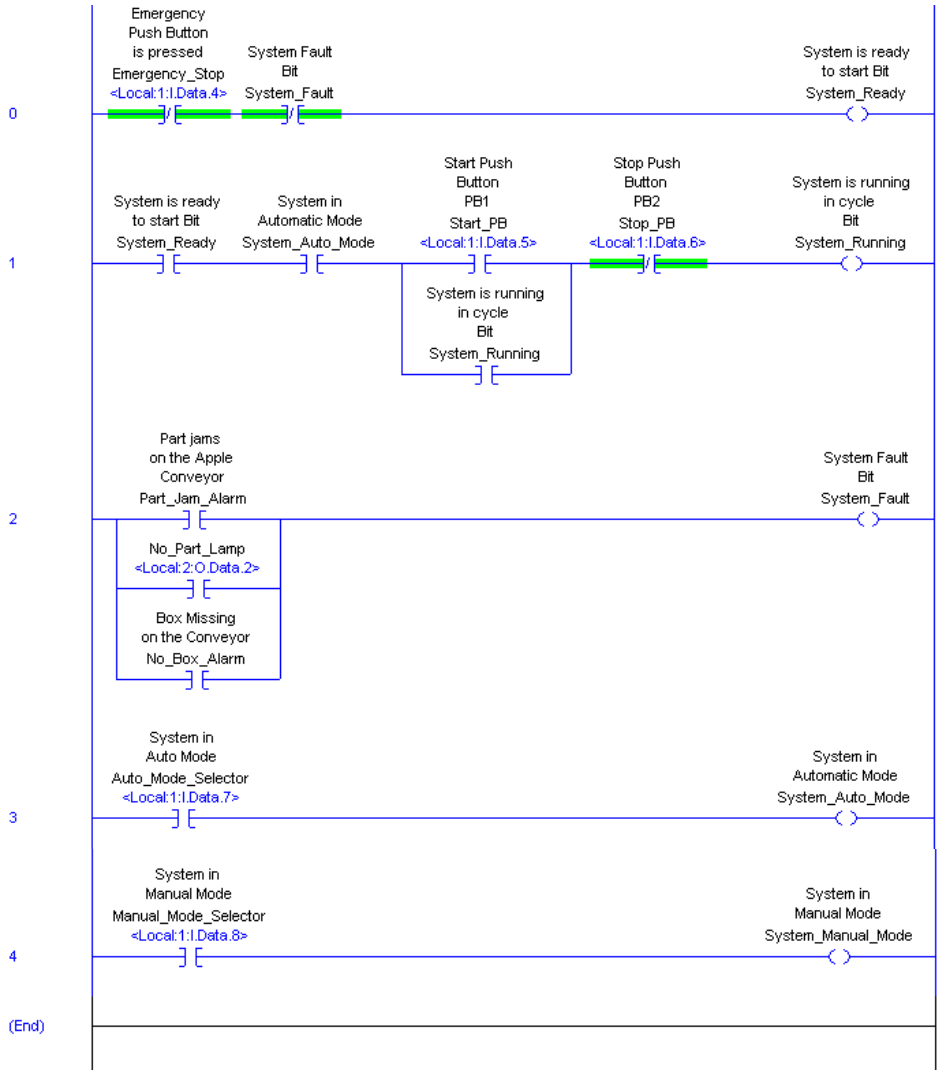
Rung 0 -> แสดงสถานะเครื่องจักรพร้อมทำงาน (Ready) เมื่อไม่มี Emergency Stop และ Fault เกิดขึ้น

Rung 1 -> แสดงสถานะเครื่องจักรกำลังทำงาน (Running) เมื่ออยู่สถานะ Ready ในโหมด Auto และกดปุ่ม Start

Rung 2 -> รวมเงื่อนไข Fault ของเครื่อง

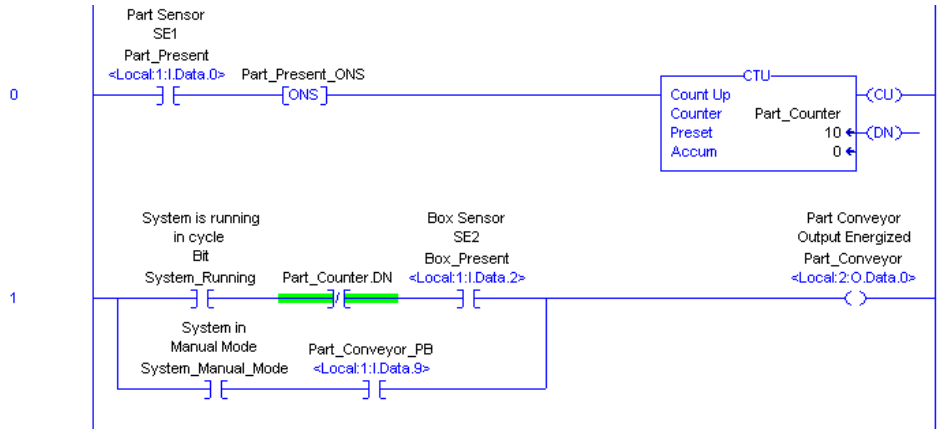
Rung 3 -> สวิตซ์เลือกโหมด Auto

Rung4 -> สวิตซ์เลือกโหมด Manual



Part_Conveyor

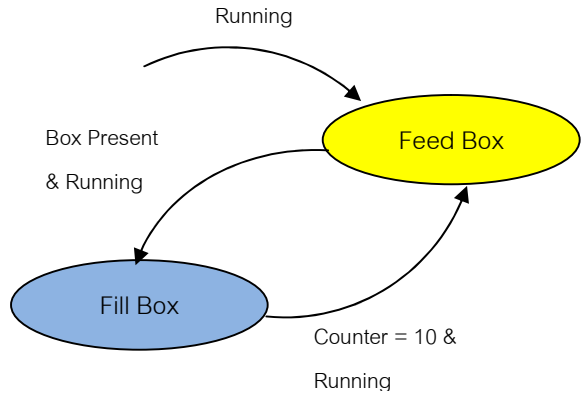
โปรแกรมแบบ Combination logic เอาท์พุตของอุปกรณ์ทำงานตามอินพุตดังนั้น Part_Conveyor ทำงานเมื่อ Part_Counter ยังนับผลแอ็ปเปิ้ลไม่ครบและกล่องยังอยู่ในตำแหน่งรับผลแอ๊บเปิ้ล อย่างไรก็ตาม ถ้าเครื่องหยุดรัน สายพานต้องหยุดทำงานด้วย เราจึงต้องนำบิต System_Running มาต่ออนุกรมเข้าไป



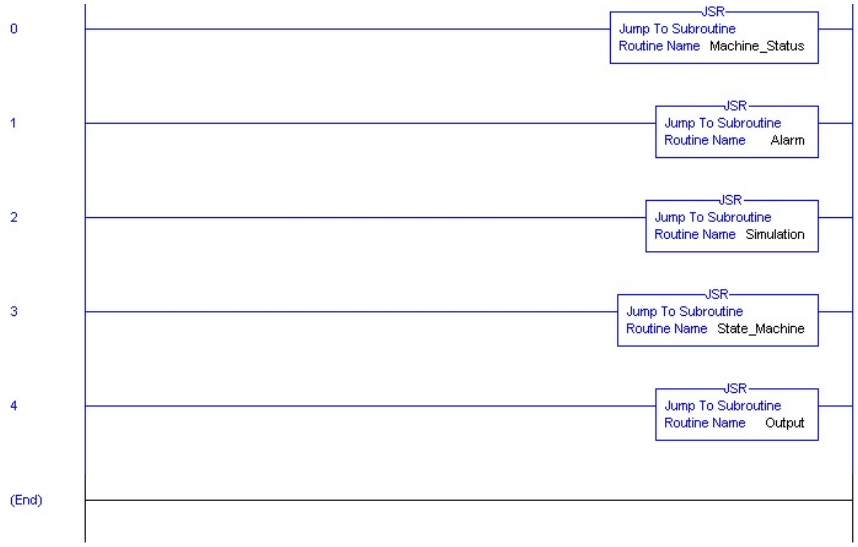
สำหรับโปรแกรมแบบ Sequential logic เอาท์พุตของอุปกรณ์จะทำงานตามอินพุตและ State (หรือ Step) ซึ่งแบ่งโครงสร้างโปรแกรมออกเป็นส่วนย่อยๆ โดยใช้ Subroutine ได้ดังนี้

- MainRoutine เป็นรูทีนหลักใช้ Jump ไป Subroutine
- Alarm เป็นรูทีนสำหรับเขียนเงื่อนไข Alarm ของเครื่อง
- Machine_Status ใช้แสดงสถานะของเครื่อง ได้แก่ Ready, Running, Fault, Auto และ Manual
- Output ใช้ Step ไปสั่งงาน Output ของอุปกรณ์ต่างๆ
- State_Machine ใช้กำหนดขั้นตอน(Step) และเงื่อนการเปลี่ยน Step ของเครื่องจักร (Transition) ตามตัวอย่างมีอยู่ 2 ขั้นตอนได้แก่ ขั้นตอนการเลื่อนกล่อง(Feed Box)และขั้นตอนการป้อนผลแอ็ปเปิ้ลลงกล่อง (Fill Part) ดัง State diagram ด้านล่าง

  สังเกตุว่าเครื่องจักรเข้าสู่ขั้นตอน Feed Box ได้นั้น ระบบต้องอยู่ในสถานะ Running และเมื่อเข้าสู่ขั้นตอน Feed Box แล้วเราจะนำ Tag ของ Step ไปสั่งงาน Output เพื่อขับสายพานของกล่อง จากนั้นจึงเปลี่ยนเข้าขั้นตอน Fill Part เมื่อเซ็นเซอร์ SE2 เช็คว่ามีกล่องอยู่ในตำแหน่งแล้ว ในขั้นตอน Fill Part เราจะนำ Tag ของ Step ไปสั่งงาน Output เพื่อขับสายพานป้อนผลแอ็ปเปิ้ล และเมื่อ Counter นับผลผลแอ็ปเปิ้ลได้ตามที่กำหนดไว้ เครื่องจะกลับเริ่มต้นที่ขั้นตอน Feed Box อีกครั้ง สลับกันไปเรื่อยๆ อย่างไรก็ตามการเปลี่ยนจาก Step หนึ่งไปยังอีก Step ต้องเปลี่ยนทีละหนึ่ง Step และอยู่ในสถานะ Running เท่านั้น



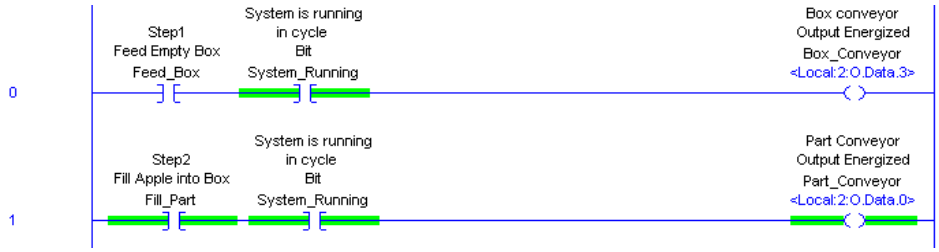
MainRoutine



Output

Rung 0 -> ถ้าอยู่ในขั้นตอนป้อนกล่องและเครื่องมีสถานะ Running ให้สั่งให้ Box Conveyor ทำงาน

Rung 1 -> ถ้าอยู่ในขั้นตอนป้อนแอ๊บเปิ้ลและเครื่องมีสถานะ Running ให้สั่งให้ Apple Conveyor ทำงาน

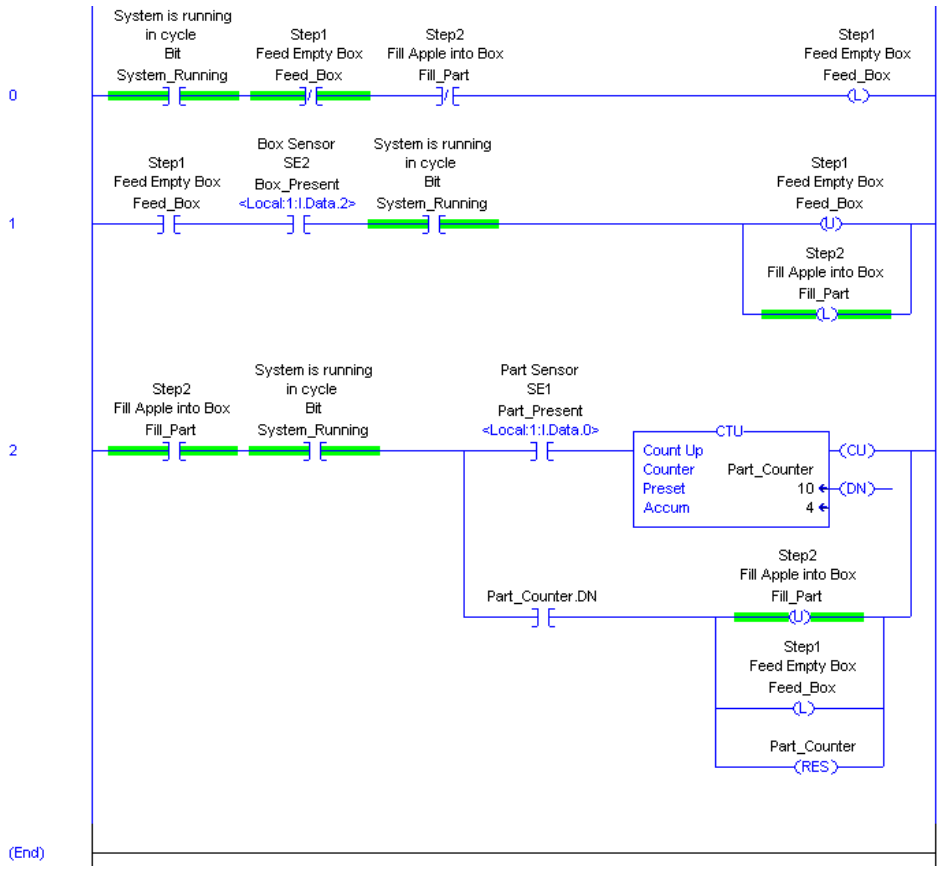


State_Machine

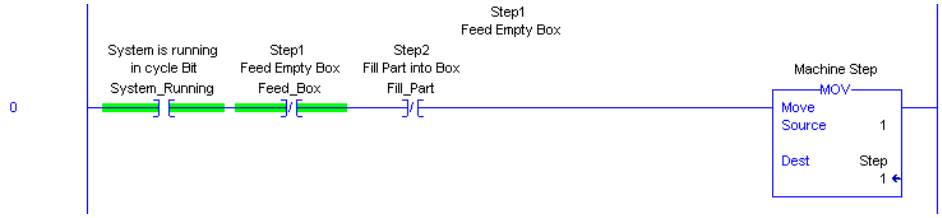
ใช้กำหนดขั้นตอน(Step)และเงื่อนไขการเปลี่ยน Step (Transition) สถานะของแต่ละ Step ถูกแทนด้วยบิต (BOOL) หรือตัวเลขจำนวนเต็ม (Integer) ก็ได้ ดังนั้นรูปแบบการเขียน State ทั่วไปมีอยู่ 2 วิธีคือ

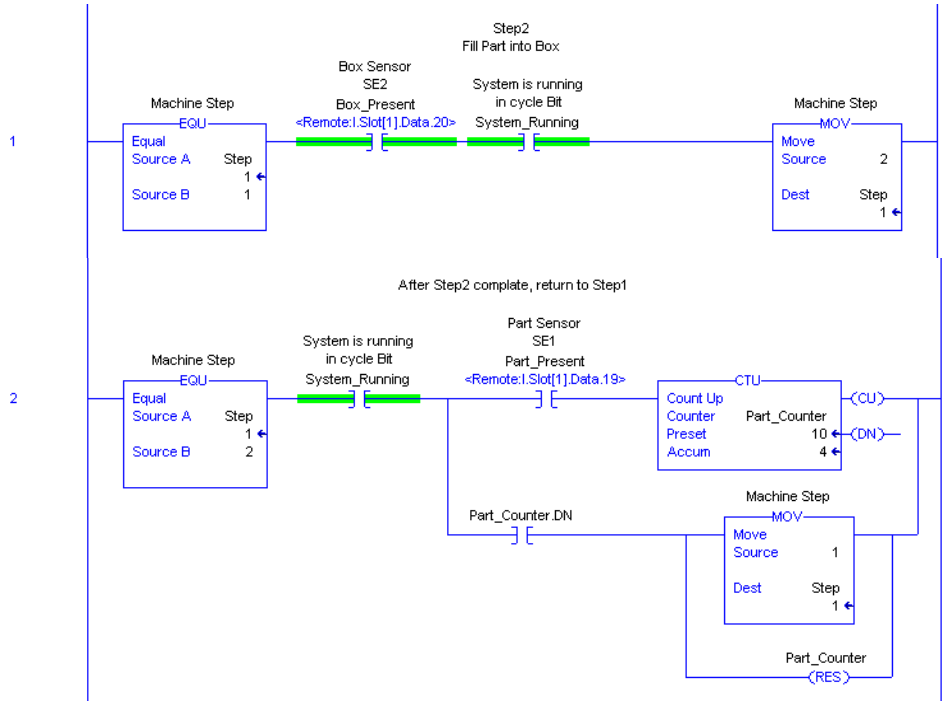
1) Latch-Unlatch ใช้กับ Step แบบบิต
2) Move ใช้กับ Step แบบจำนวนเต็ม

แบบ Latch-Unlatch (Bool)



แบบ Move (Integer)





อย่างไรก็ตาม เรามักนำใช้วิธีการเขียนโปรแกรมทั้งสองแบบมาใช้ร่วมกัน เพราะถ้าใช้ State machine เพียงอย่างเดียว เราต้องเขียน State ให้ครอบคลุมทุกๆเงื่อนไข ซึ่งมีจำนวนมากและวุ่นวายพอควร ดังนั้นถ้าส่วนไหนมีการทำงานไม่ซับซ้อนก็เขียนโปรแกรมแบบ Combination แต่ถ้าส่วนไหนมีลำดับขั้นตอนที่แน่นอนก็เขียนแบบ State machine ช่วย