Lab 1 - Quick solution:

The intent of this program is to demonstate a simple Automatic event driven PLC program based on the Assembler Video provided for FactoryIO.

Rung 0

| Cycle_Start_PB | Auto_PB | Cycle_Stop_PB | Factory_Running | E_Stop | | Cycle_Running_Lt |
| <BOOL_IN_10> | <BOOL_IN_14> | <BOOL_IN_12> | <BOOL_IN_15> | <BOOL_IN_13> | | <BOOL_OUT_9> |

Cycle_Running_Lt
<BOOL_OUT_9>

Run the Lid Feed Conveyor until the Lid is in place to be Clamped.
Run the Base Feed Conveyor until the Base is in place to be Clampled.
Restart both conveyors once the Lid in Place Timer has completed (Assembly Complete).

Rung 1

Cycle_Running_Lt
<BOOL_OUT_9>   Lid_At_Place_Dwell.DN   Lids_Conveyor
                                                               <BOOL_OUT_3>

Rung 2

Cycle_Running_Lt
<BOOL_OUT_9>   Base_At_Place_Dwell.DN   Base_Conveyor
                                                               <BOOL_OUT_6>

Rung 3

Cycle_Running_Lt   Lid_At_Place
<BOOL_OUT_9>   <BOOL_IN_3>   Lid_Placed

                     Lid_At_Place_Dwell.DN

TON	
Timer	Lid_At_Place_Dwell	(EN)
Preset	600	(DN)
Accum	0	

Rung 4

Cycle_Running_Lt   Bases_at_Place
<BOOL_OUT_9>   <BOOL_IN_6>   Lid_Placed

                     Base_At_Place_Dwell.DN

TON	
Timer	Base_At_Place_Dwell	(EN)
Preset	600	(DN)
Accum	0	

Turn on the Suction to Grab the Lid once the input "Item Detected" turns ON.

Rung 5

Item_Detected
<BOOL_IN_2>   Lid_Placed   Grab
                                                               <BOOL_OUT_2>

Both Lid and Base Clamps have the condition to close once thier Lid or Base is in place and them to Unclamp once an Assembly has been built.
The Lid Clamp has an added condition to Unclamp once a Lid has been Grabbed.

Rung 6

                        Grab
Lid_At_Place_Dwell.DN   <BOOL_OUT_2>   Clamp_Lids
                                                               <BOOL_OUT_4>

Rung 7

Base_At_Place_Dwell.DN   Clamp_Bases
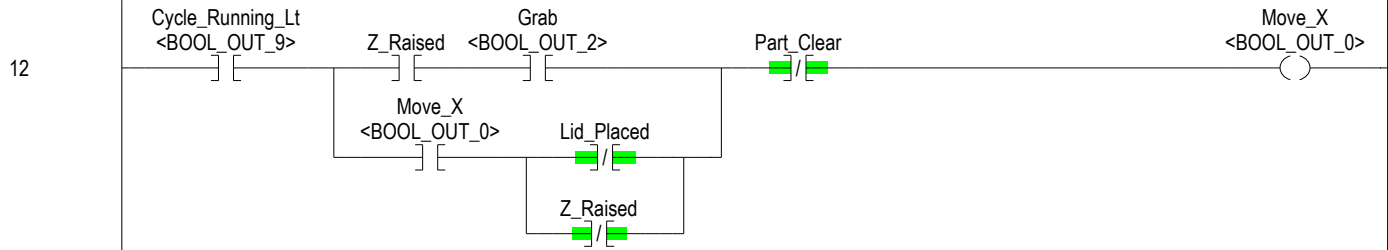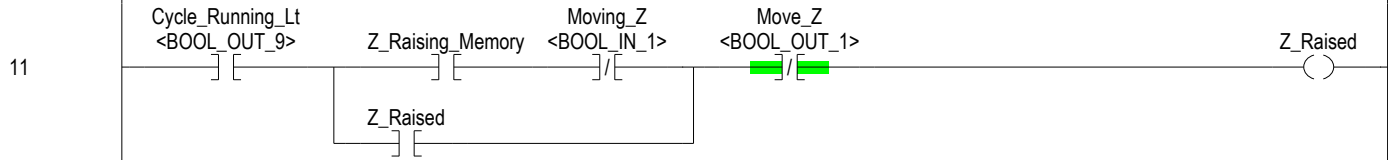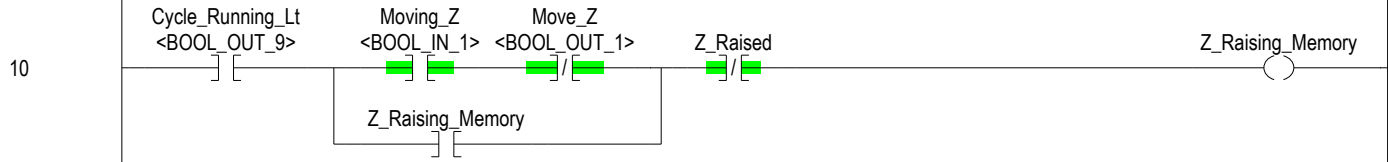   <BOOL_OUT_7>

Advance_X_Dwell is needed to account for the delay between issuing a command and getting feedback. Since there is no Advanced Sensor and Move_X must stay on to keep the X-Axis in the Advanced Position, the Dwell and NOT Moving_X can be used in place of Advanced instead of creating a specific Bit.
.

Rung 8

Move_X
<BOOL_OUT_0>
┤ ├

TON
Timer Advance_X_Dwell (EN)
Preset 200 (DN)
Accum 0

Rung 9

Cycle_Running_Lt   Lid_Clamped   Base_Clamped   Grab   Move_Z
<BOOL_OUT_9>   <BOOL_IN_4>   <BOOL_IN_7>   <BOOL_OUT_2>   <BOOL_OUT_1>
┤ ├   ┤ ├   ┤ ├   ┤/├   ( )

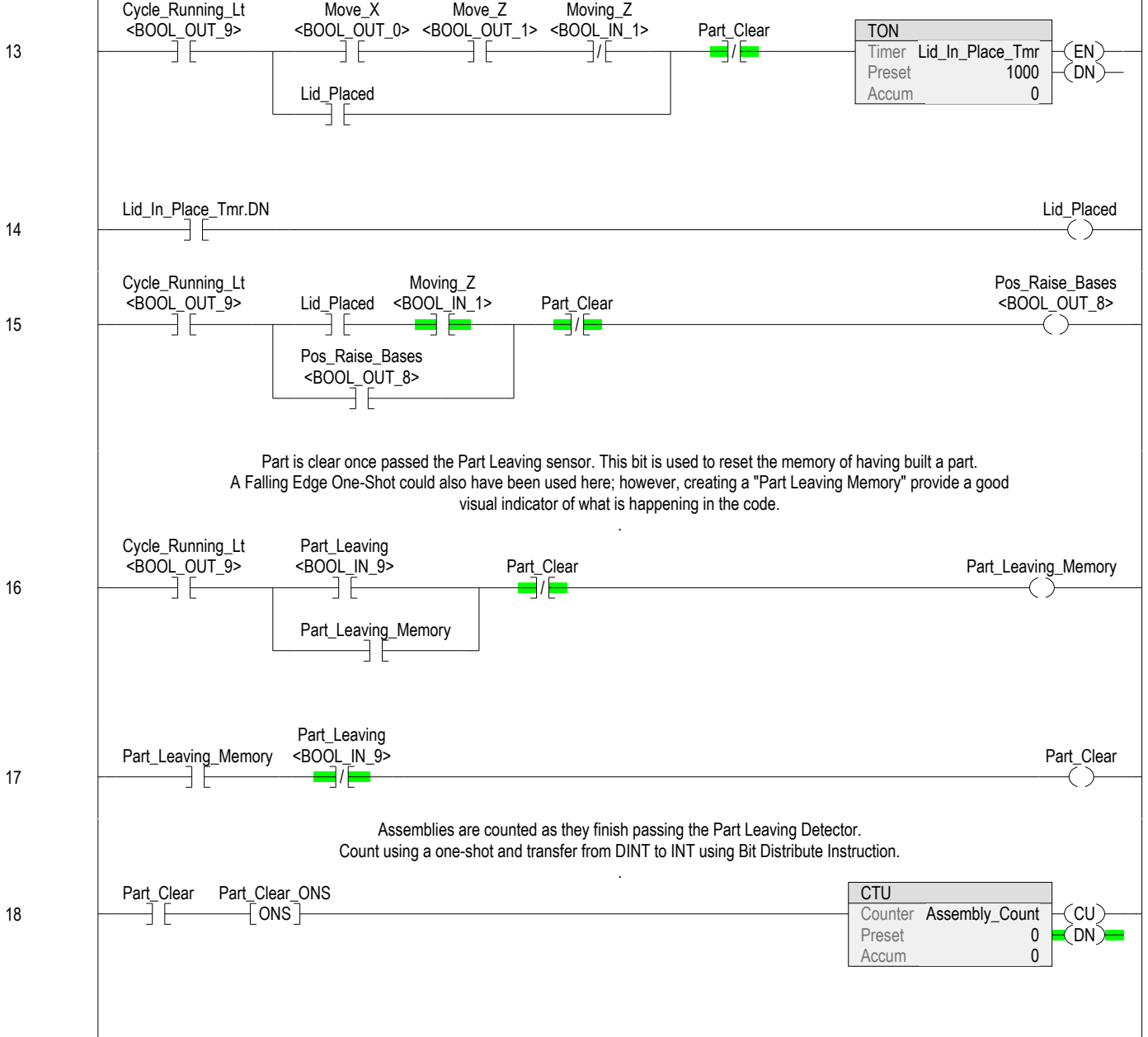Advance_X_Dwell.DN   Moving_X <BOOL_IN_0>   Lid_Placed
┤ ├   ┤/├   ┤/├
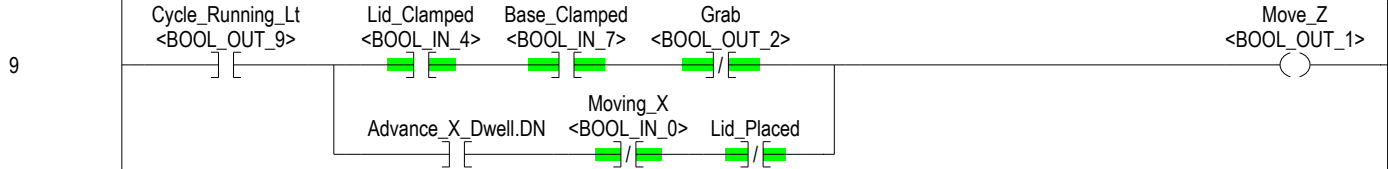
There is no limit switch for the Z-Axis and we need to ensure the Z-Axis is fully raised before Advancing X.
Using the Move Command and Moving Feedback, Z_Raising_Memory is Sealed in. When the Axis stopped moving, Raised is turned ON and Sealed in at the same time, releasing the seal on the Raising Memory Bit. Note that the technique used for the Axis Motion above could have worked as well. This just demonstrate an alternate method (there is no "correct" method, however, unique bits would be more structured - See LAB 2) The Axis responds same as controls using a Single-Ended-Spring-Return Solenoid. Turning OFF Move_Z Raises the Cylinder.
.

Rung 10

Cycle_Running_Lt   Moving_Z   Move_Z   Z_Raised   Z_Raising_Memory
<BOOL_OUT_9>   <BOOL_IN_1>   <BOOL_OUT_1>   Z_Raised   ( )
┤ ├   ┤ ├   ┤/├   ┤/├

Z_Raising_Memory
┤ ├

Rung 11

Cycle_Running_Lt   Moving_Z   Move_Z   Z_Raised
<BOOL_OUT_9>   Z_Raising_Memory   <BOOL_IN_1>   <BOOL_OUT_1>   ( )
┤ ├   ┤ ├   ┤/├   ┤ ├

Z_Raised
┤ ├

Rung 12

Cycle_Running_Lt   Grab   Move_X
<BOOL_OUT_9>   Z_Raised   <BOOL_OUT_2>   Part_Clear   <BOOL_OUT_0>
┤ ├   ┤ ├   ┤ ├   ┤/├   ( )

Move_X
<BOOL_OUT_0>   Lid_Placed
┤ ├   ┤/├

Z_Raised
┤/├

This is a variation of how to detect the end of a motion using the Move Command the Moving Feedback.
Instead of creating a memory that Z is Advancing, Move_Z ON and Moving_Z OFF are conditions to detect end of stroke. However, since the Moving Status of the Axis does not respond immeidately to the Move Command, there is a short period of time a the begining the stroke where the Move Command is ON and the Axis Moving Input has not turned on.
Although not technically a RACE condition, a timer can be utilized to ignore the overlap. On a real machine, the Returned Limit Switch woudl still be ON, for example, during a short peroid of time after Solenoid Turns on and the cylinder
actually starts to move. Good "Real World" discussion.
.

Rung 13

Cycle_Running_Lt <BOOL_OUT_9> | Move_X <BOOL_OUT_0> | Move_Z <BOOL_OUT_1> | Moving_Z <BOOL_IN_1> (NC) | Part_Clear (NC)

Lid_Placed (parallel branch)

TON
Timer: Lid_In_Place_Tmr — (EN)
Preset: 1000 — (DN)
Accum: 0

Rung 14

Lid_In_Place_Tmr.DN ——— Lid_Placed ( )

Rung 15

Cycle_Running_Lt <BOOL_OUT_9> | Lid_Placed | Moving_Z <BOOL_IN_1> | Part_Clear (NC) ——— Pos_Raise_Bases <BOOL_OUT_8> ( )

Pos_Raise_Bases <BOOL_OUT_8> (parallel branch)

Part is clear once passed the Part Leaving sensor. This bit is used to reset the memory of having built a part.
A Falling Edge One-Shot could also have been used here; however, creating a "Part Leaving Memory" provide a good visual indicator of what is happening in the code.
.

Rung 16

Cycle_Running_Lt <BOOL_OUT_9> | Part_Leaving <BOOL_IN_9> | Part_Clear (NC) ——— Part_Leaving_Memory ( )

Part_Leaving_Memory (parallel branch)

Rung 17

Part_Leaving_Memory | Part_Leaving <BOOL_IN_9> (NC) ——— Part_Clear ( )

Assemblies are counted as they finish passing the Part Leaving Detector.
Count using a one-shot and transfer from DINT to INT using Bit Distribute Instruction.
.

Rung 18

Part_Clear | Part_Clear_ONS [ONS]

CTU
Counter: Assembly_Count — (CU)
Preset: 0 — (DN)
Accum: 0

19

BTD	
Source	Assembly_Count.ACC
	0
Source Bit	0
Dest	Part_Count
	<INT_OUT_0>
	0
Dest Bit	0
Length	16

20

Factory_Running
<BOOL_IN_15>
—]/[—

Assembly_Count
—(RES)—

(End)