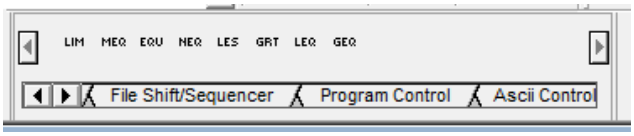
EET165 Lecture #8

1) **Review:** Answer any questions from last week.

2) **Control Instructions:** The control instructions are commands that activate or deactivate different parts. All of these instructions are found under the ***Program Control*** tab.



The control instructions covered in this lecture are:

   a. **MCR / MCR** – MCR stands for Master Control Reset. This will disable or enable a block of code. The first MCR command is the "start fence" and the second MCR command is the "end fence". When the Start Fence is true, the Master Control Reset is activated and the code is ignored. All of the non-retentive outputs are cleared and the retentive outputs do not change.

   b. **JMP / LBL** – JMP stands for Jump and LBL stands for label. This will jump over a piece of code and "freeze" the existing outputs. It does not clear anything, it just jumps over the code as if it was not there.

   c. **JSR / SBR (and RET)** – JSR is jump to subroutine, SBR is the start of the subroutine, and RET is return from the subroutine. Subroutines are pieces of code that can be skipped or activated in a similar way as the JMP and LBL. But the subroutines can be called from multiple places in the code.

3) **MCR/ MCR:** The MCR command is used in pairs. They are like a light switch. The first one turns off the code; the next one turns it back on. Every time you encounter an MCR it toggles on and off. Some of the members of industry around here told us that they do not use this command and try to eliminate them when they can. They don't like this command for a few reasons. You should understand how the MCR works, but we will not be using them.

   a. **TOF Confusion:** If a TOF is inside the MRC there is a problem. An MRC resets all the values to zero. That includes the input to a TOF – and that will start the TOF counting.

   b. **MCR is not a Master Relay:** Some people think this is the same as a master relay. A master relay disconnects power from the system. A master relay stops electrocution and sparks. A Master Control Reset just resets the logical values – it will not stop sparks or save a life.

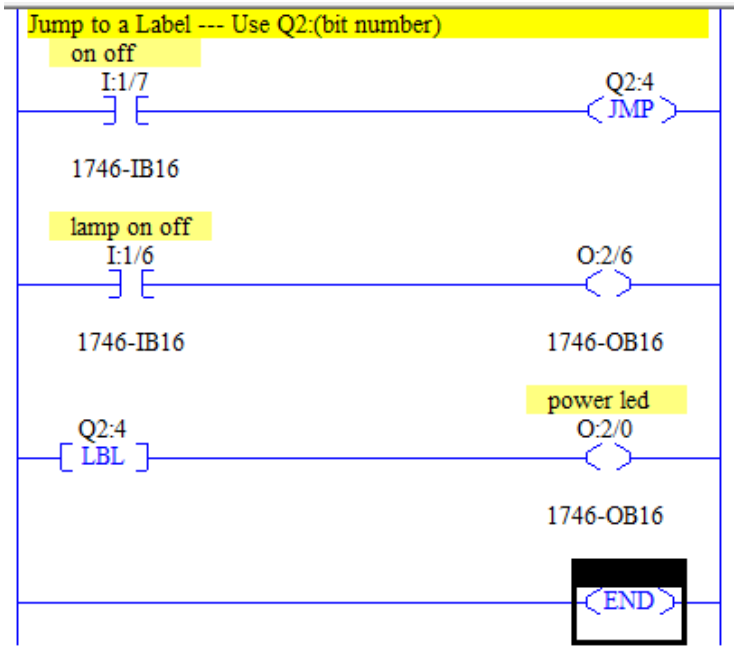
4) **JMP / LBL:** JMP and LBL also disable part of the code. But it does not mess with the states of the inputs. It just "freezes" them in their current state. This is a bit more safe because you won't have unexpected results.

Note 1: A LBL can't be on its own line, it needs an output. In this example, a bit is used as the output, but the bit is not used for anything.

Note 2: You can jump forward OR backward. This means you can jump to any rung, before or after the JMP rung. However, if you jump before the JMP command, you might go into a loop that lasts longer than the PCL program will allow. The PLC might "time-out" and shut down (or Reset) the program. It is a good idea to only jump below the JMP command.

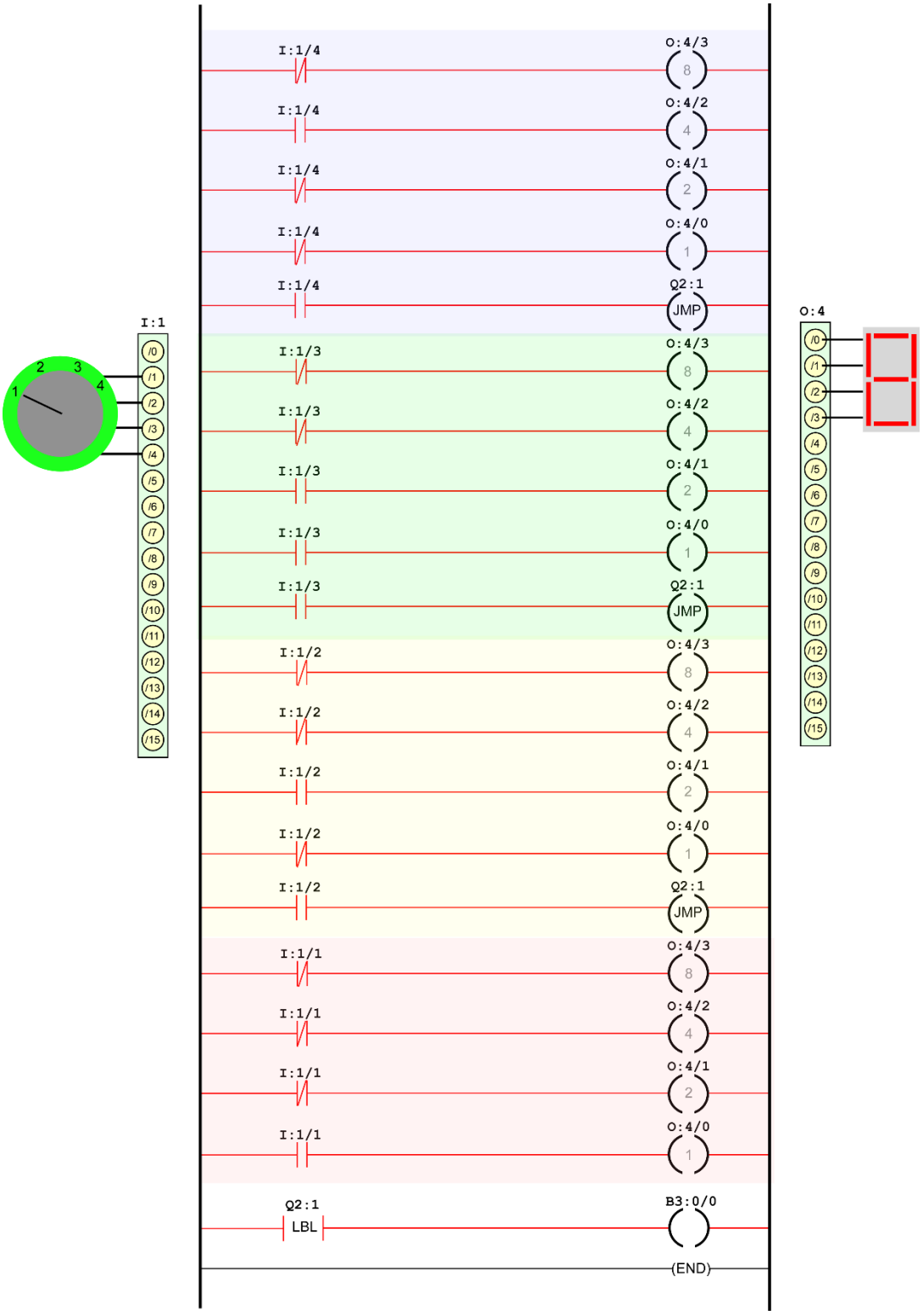
Note 3: Never jump into a MCR block, it will turn it on, even if it is off. Don't do it!

a. **Example 1:** The jump and label need to match. The id is Q2:(Bit number), for example Q2:4.



When I:1/7 is on, the next rung is skipped. That means if I:1/6 is turned on, O:2/6 will not turn on.

b. **Example 2:** In this example, the selector switch will pull the selected number high and the rest of the inputs low. The output will take in a 4-bit binary value (BDC) and display the binary value as a base 10 digit.



5) **JSR / SBR (and RET)** – JSR jumps to a subroutine (SBR). If you want to quit out of a subroutine early, you can use the RET command. If you want to run the entire subroutine, you do not need the RET command. The end will work just like a RET. Subroutines are useful for many reasons.

   a. **Speed:** A subroutine takes a chunk out of the main code and moves it to a different file. That means the main code will have fewer rungs and run faster.

   b. **Compartmented:** The code is set up in compartments. This lets you focus on a small piece of code and getting it working. Once working, you can forget about it. If you need to update it later, you can go directly to the subroutine, and it will be easy to find. You can then focus on the code without any other code distracting you.

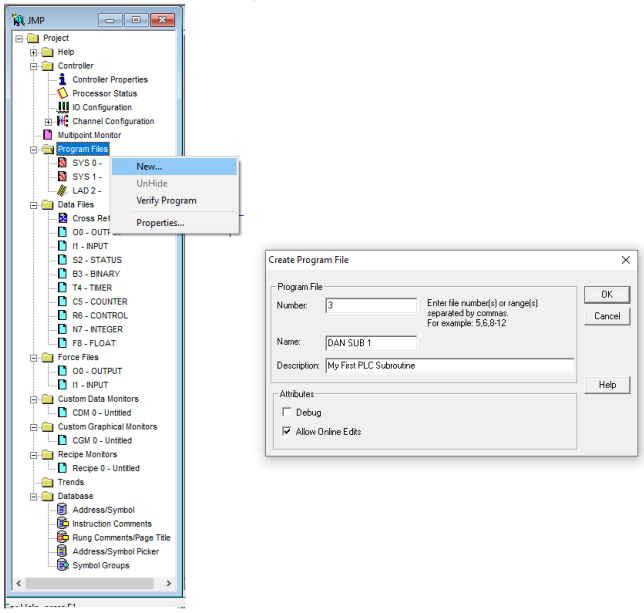
   c. **Reuse:** The subroutines can be reused many times. If code needs to be activated in many places (such as an alarm that alerts the people to a problem) it can be written once and used many times. This makes the overall code a lot shorter. If you end up changing the alarm system, you only need to update it in one place.

   d. **Recipe:** Subroutines can be used to select a "Recipe". A piece of equipment can be used to produce more than one type of item. The control procedure for each item is called a recipe. To avoid having to load and reload new programs each time a new item is produced, each unique code is loaded into a subroutine, and the subroutine is selected with a selector switch or other input.

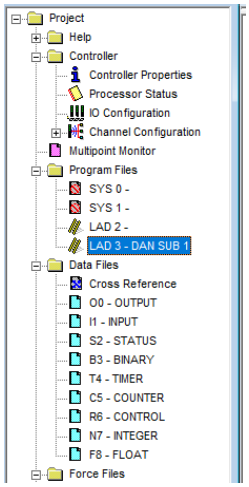
   e. **RET:** RET stands for Return. This command will jump back to the place it came from. This is automatically called when you hit the END of the subroutine. You can return at any point, but I don't like doing that. If the subroutine has some parts that run and some that don't, I would just make it into 2 different subroutines. I don't often use the RET command.

6) **How to create a subroutine:** A subroutine will be added to the program folder. The first two files are SYS 0 and SYS 1. The third file is the main PLC program, it has the name LAD 2, this is the file that you write when you open a new file. The subroutines are any other LAD files that are not in use. In this example, the subroutine will be LAD 3.

To open the subroutine, right-click on the program files and select "New". This will bring up a window that will let you choose what LAD file you want to open (in this case 3). You can also give the file a "nick-name" to help you identify what file is doing what job.



This will create a new Ladder Logic Program called LAD 3. This procedure can be repeated to create as many different Ladder Logic Programs (PLC programs) as you wish.



7) **Example:** In the example below, when I:1/7 is high (powered, on, true, or however you think of it) then the code in file LAD 3 is called because the JSR is jumping to U:3. LAD 3 will check to see if I:1/7 is on and if the second timer is NOT done, then Timer 1 will start to run. While the program is running (T4:1/TT) the power LED is on. When it is finished running the TT signal goes low (unpowered, off, false, or however you think of it). This will turn Off the power LED. The done signal will go high (T4:1/DN) and this will start the second timer (T4:2). When timer 2 finishes, the done bit (T4:2/DN) goes high. This will reset Timer 1 and timer 1 will reset timer 2. When I:1/7 is off, the power LED O:2/0 will not be lit.