PMAC PLC Programming Techniques

PMAC’s PLC programs are used to accomplish a wide variety of tasks. These tasks range from making simple decisions to completely customizing the behavior of PMAC. Although these tasks vary greatly in scope and complexity, the programming structures used to accomplish them are relatively few in number. These structures are simple to learn and to use. Once mastered they can be combined like building blocks to create a complex program out of very simple and easy to debug pieces.

There are two steps to building a PLC program. One is selecting the correct logical structures. The other is combining them in the correct manner.

Logical Building Blocks

These are the small pieces of PLC code used to do very simple parts of the PLC.

Counter Delay

If wanting a small delay after a PLC commands an action or sets an I/O, a simple counter can be used. Be aware of the risks. Counters work at the speed of the PMAC CPU and therefore are not of a fixed time. If the delay time needs to be a fixed time or portable to different PMAC CPU speeds or firmware versions, a Timer Delay is a better option.

Here is the logical structure of a counter delay and some sample PMAC code.

![Counter Delay Diagram]

Timer Delay

To have a small delay after a PLC commands an action or sets an I/O, a Timer Delay is a good option when the delay is always at least X msec. Timers work independent of the speed of the PMAC CPU and are therefore portable to different PMAC CPU speeds or firmware versions. A Timer Delay is a better option than a Counter Delay. If the time for the commanded I/O or action to actually take place can vary greatly then a Conditional Delay could be a better option.

To write this in PMAC code, first point an M-Variable to a timer register. These are found in the memory map in the PMAC Manual.

```
M10->X:$0700,0,24,S
```

The PMAC code would be as follows for a one second delay.

```
M10=1000*8388607/I10
WHILE
(M10 > 0)
ENDW
```
In this case, the desired action was to set M10 to a value. More often than not, there will be no desired action during the WHILE loop.

**Conditional Delay**

For a small delay after a PLC commands an action or sets an I/O, a Conditional Delay is a good option when the delay time varies due to external factors. Conditional Delays work independent of the speed of the PMAC CPU and are therefore portable to different PMAC CPU speeds or firmware versions. A Conditional Delay works using a feedback principle. When action or I/O is commanded the PLC waits for another event which means the action actually happened.

To use a Conditional Delay, therefore there must be an event that can be used as the feedback. Usually, this event is either an I/O or an internal PMAC Status bit. When a Conditional Delay is used, consider what happens if the condition never happens. Should it timeout of the condition or stay locked in it forever?

To write this, setup two M-variables, one for a timer and another for the condition. Choose the Desired Velocity Zero motor status bit for the condition. Then process the setup.

M10$=X:0700,0,24,S
M133$=X:003D,13,1
**Toggle Switch I/O**

For this discussion, the definition of Toggle Switch I/O means I/O that has one action for its ON state and one for its OFF state. This is in contrast to Push Button I/O which does one action the first time it goes into the ON state and then a different action the next time it goes into its ON state.

Toggle Switch I/O is the most common type of I/O. It is used for most types of operator panel I/O.

To write this, set up an M-Variable for the I/O and reserve a variable for the latch. Assume machine input one and use M11 to read its state. For the latch, use P1.

```
M11->Y:$FFC2,0,1
```

Now the code and logical structure are as follows:

To make this realistic, command a motor to JOG when the input is ON and to STOP when it is OFF.

```
IF(M11=1 AND P1=0) P1=1
  CMD"#1J+" ELSE
  IF(M11=0 AND P1!=0) P1=0
    CMD"#1J/"
ENDIF
ENDIF
```

The statement `P1!=0` was used in place of `P1=1`. This is to make sure that regardless of the state of P1 when PMAC powers on this code will function.

**Push Button I/O**

For this discussion, the definition of Push Button I/O means I/O that has one action the first time it goes into the ON state and then a different action the next time it goes into its ON state. This is in contrast to Toggle Switch I/O which is I/O that has one action for its ON state and one for its OFF state.

This structure is very similar to Toggle I/O. The difference being that an addition variable is needed to remember the previous command. So there is one variable to remember if the input was ON or OFF and one to remember which command was sent last.

To make this realistic, command a motor to JOG when the input is ON the first time and to STOP when it is ON the next time. The logical diagram will not be given but the flow will be described in words then in PMAC code.
IF (CONDITION = TRUE AND LAST_COMMAND != FIRST_COMMAND)
   AND (LATCH = FALSE)
   LATCH = TRUE
   LAST_COMMAND = FIRST_COMMAND
   {ACTION}
ELSE
   IF (CONDITION = TRUE AND LAST_COMMAND = FIRST_COMMAND)
      AND (LATCH = FALSE)
      LATCH = TRUE
      LAST_COMMAND = SECOND_COMMAND
      {ACTION}
   ELSE
      IF (CONDITION = FALSE AND LATCH != FALSE)
         LATCH = FALSE
         {ACTION}
      ENDIF
   ENDIF
ENDIF

In PMAC code, decide that P2 is the command memory. When P2=1, a Jog plus was performed and when P2=2, a Jog minus was performed.

IF(M11=1 AND P1=0 AND P2!=1)
   P1=1
   P2=1
   CMD"#1j+
ELSE
   IF(M11=1 AND P1=0 AND P2=1)
      P1=1
      P2=2
      CMD"#1J+
   ELSE
      IF (M11= AND P1!=0
         P1=0
      ENDIF
   ENDIF
ELSE
   IF (CONDITION = FALSE AND LATCH != FALSE)
      LATCH = FALSE
      {ACTION}
   ENDIF
ENDIF

Some people wonder why two variables must be used. It is possible to do this with only one variable. The key is to wait for the condition to go false before another action is allowed on the true state.

**Combining Structures**

Now that the building blocks are in place, how should they be combined? To decide, ask about the purpose of this program. Will it start and stop several actions that are independent of each other or should it execute sequentially to control some sort of process? Refer to independent operations as an I/O Read/Write. Examples of I/O Read/Write are an operator control panel. This panel will have switches that control jogging, coolant, etc. For a sequential process, think of homing a milling machine. First, the cutting heads are pulled up, then the axes are homed, and then it is moved to the offsets.

**Real Time I/O Read/Write**

This is the standard mode that PMAC uses. When an M-variable is pointed to an I/O location, PMAC makes this a direct link. Each time this variable is used, PMAC actually reads the physical I/O that is addressed. This method is different fundamentally from a standard PLC. A standard PLC reads all inputs at the beginning of its scan and copies their state to memory. The programs only access and work with the memorized I/O. At the end of the scan, all outputs are written to the physical hardware.

To have Real Time I/O with PMAC, first set up a M-variable to point to I/O:
M1->Y:$FFC2,0,1
M11->Y:$FFC2,8,1
Now just use these variables.

```plaintext
IF(M11=1)
  M1=1 ENDIF
IF(M11=0)
  M1=0 ENDIF
```

Each time M1 or M11 is used in the above code the state of the physical I/O point is being read. Although this is a nice capability, it can lead to confusion. In the above code, assume that the first IF condition or the second will be entered and never both on the same PLC scan. This is a faulty assumption. Since the state of the input pin can change at any time, it can change right after M11 is read for the first IF. Both IF conditions are entered during a single PLC scan. In this example, it would not matter if this happened. However, in other cases it could matter.

**One Time I/O Read/Write**

This is not the standard mode used by PMAC. This method is what a standard PLC uses. A standard PLC reads all inputs at the beginning of its scan and copies their state to memory. The programs only access and work with the memorized I/O. At the end of the scan, all outputs are written to the physical hardware.

Using One Time I/O Read/Write requires more setup from the programmer. The benefits are more robust and it is easier to debug code.

The logical structure for this type of programming is like this:

The setup required for this type of programming consists of reserving memory locations to store to memorized I/O values. For the following example, use P1 and P11 to do the same program as above.

Again our real I/O is:

```plaintext
M1->Y:$FFC2,0,1  
M11->Y:$FFC2,8,1
```

And the PMAC code is:

```plaintext
IF(P11!=M11)
  P11=M11
  IF(P11=1)
    P1=1
  ENDIF
  IF(P11=0)
    P1=0
  ENDIF
  M1=P1
ENDIF
```

Now the state of P11 will be constant for the entire scan of this PLC.
Process Control

Another way to think of Process Control is to think of a State Machine. A machine that always starts at state 1 and cannot advance to the next state until some condition is satisfied. It is a combination of Conditional Delays.

To combine Conditional Delays into a state machine, reserve a variable to store the value of the current state. This is not required but is very useful for debugging and monitoring the process. In addition, the power of state machine programming is the ease of debugging it affords. The logical flow of a three state system follows:

```
IF(START_CONDITION = TRUE)
STATE = 1
{actions}
WHILE(STATE = 1)
{actions}
    IF(EXIT_STATE_1_CONDITION = TRUE)
        STATE = 2
    {actions}
ENDIF ENDW
{actions}
WHILE(STATE = 2)
{actions}
    IF(EXIT_STATE_2_CONDITION = TRUE)
        STATE = 3
    {actions}
ENDIF ENDW
{actions}
WHILE(STATE = 3)
{actions}
    IF(EXIT_STATE_3_CONDITION = TRUE)
        STATE = 4
    {actions}
ENDIF ENDW
{actions}
ENDIF
```

Another nice feature of a state machine is that although the code for the process can be very large and complex. Only a small part is being scanned at any given time so it is very efficient.

For a real PMAC example lets take the situation of homing a machine. The machine must home motors 1, 2, and 3 when machine input 1 is pressed. Motor 3 must be homed first. Then motors 1 and 2 can home. Then all motors must be moved to an offset location. The configuration of the machine has the home sensors at the Negative end of travel. This means that the +LIM pin (see PMAC manual for details) must be used to see if it is past the home switch.

To do this, use P1 as the state counter and also the following M-variable definitions.

```
M11->Y:$FFC2,0,1   ; MACHINE INPUT 1
M120->X:$C000,20,1 ; #1 HMFL STATUS
M122->X:$C000,22,1 ; #1+LIM STATUS
M133->X:$003D,13,1 ; #1 DESIRED VELOCITY ZERO
M145->Y:$0814,10,1 ; #1 HOME COMPLETE
M220->X:$C004,20,1 ; #2 HMFL STATUS
M222->X:$C004,22,1 ; #2+LIM STATUS
M233->X:$0079,13,1 ; #1 DESIRED VELOCITY ZERO
M245->Y:$08D4,10,1 ; #2 HOME COMPLETE
M320->X:$C008,20,1 ; #3 HMFL STATUS
M322->X:$C008,22,1 ; #3+LIM STATUS
M333->X:$00B5,13,1 ; #1 DESIRED VELOCITY ZERO
M345->Y:$0994,10,1 ; #1 HOME COMPLETE
```
With this setup, use the following PMAC code:

```plaintext
IF(M11=1) ; start signal given
    P1=1 ; set state
    WHILE(P1=1)
        IF(M322=1 OR M320=1) ; past mark?
            CMD"#3J:100" ; move off sensor
            WHILE(M333=1) ; wait for start
                ENDW
            WHILE(M333=0) ; wait for end of move
                ENDW
            ENDIF
        P1=2 ; state finished
        ENDW
M345=0 ; clear flag
    CMD"#3HM" ; do home move
    WHILE(P1=2)
        IF(M345=1 AND M333=1); wait for motion stopped
            P1=3 ; state finished
        ENDIF
    ENDW
    WHILE(P1=3)
        IF(M122=1 OR M230=1) ; past mark?
            CMD"#1J:100" ; move off sensor
            WHILE(M133=1) ; wait for start
                ENDW
            WHILE(M133=0) ; wait for end of move
                ENDW
            ENDIF
        IF(M222=1 OR M220=1) ; past mark?
            CMD"#2J:100" ; move off sensor
            WHILE(M233=1) ; wait for start
                ENDW
            WHILE(M233=0) ; wait for end of move
                ENDW
            ENDIF
        P1=4 ; state finished
    ENDW
    M145=0 ; clear flag
    M245=0 ; clear flag
    CMD"#1HM #2HM" ; do home move
    WHILE(P1=4)
        IF(M145=1 AND M133=1); wait for motion stopped
            AND(M245=1 AND M233=1)
            P1=5 ; state finished
        ENDIF
    ENDW
    CMD"#1=100 #2J=200 #3J=00"; move to offsets
    WHILE(P1 = 5)
        IF(M133 = 0 OR M233 = 0 OR M333 = 0) ; motion began?
            P1 = 6 ; state finished
        ENDIF
    ENDW
    WHILE(P1 = 6)
        IF(M133 = 1 AND M233 = 1 AND M333 = 1) ; motion stopped?
            P1 = 7 ; state finished
        ENDIF
    ENDW
    ENDW
```
Control Panel PLC

This example shows how to use One Time I/O Read/Write to emulate the functions of PMAC’s J2 connector. It combines several of the programming techniques discussed above into a real world example. It also uses the PMAC Executive feature of Macro programming. This allows the programmer to use meaningful phases in substitution for PMAC variables.

Macro programming also makes the programs more portable. Once the program is functioning, it does not need to change to use a different variable for some feature. Only the Macro definition needs to change.

="/*********************** CTRL PANEL **************************
// Many people have trouble programming PMAC PLC’s because of the real-time
// nature of PMAC-M-variables. Usually M-variables point to inputs and
// outputs. Each time a M-variable is used the actual hardware line is read
// and the current value of the input or output is used. In some circumstances
// this is a very desirable feature, but it can also cause logical problems
// for programmers who are not expecting this.
//
// Many times PLC logic is written with the idea that the I/O values
// stay constant during the PLC scan. This is the standard method
// used in Hardware PLC’s. When the I/O // is constant during a PLC
// scan the logic for the PLC is easier.
//
// This PLC is an example of robust PLC programming. It uses the method of
// combining several real-time inputs into a static word which can then
// be easily dealt with. This PLC was written to simulate the functions
// of the J2 (control panel) connector but any I/O tasks could be performed.
//
// PMACs other than 1 and 1.5 do not dedicate the J2 connector to control
// panel use. If you would like to simulate these features then this PLC
// can be used. Just pick 12 inputs from any PMAC connector and redefine
// the Macro names below. Download this PLC and you will have the same
// functionality as the dedicated J2 connector.
//********************************************************************************
CLOSE
ENDG
DEL GAT
#define INP 01 M M20
#define INP_01_ADR Y:$FFC0,8,1 ; Jog Minus Input
#define INP_02_M M21
#define INP_02_ADR Y:$FFC0,9,1 ; Jog Plus Input
#define INP_03_M M22
#define INP_03_ADR Y:$FFC0,10,1 ; Prejog Input
#define INP_04_M M23
#define INP_04_ADR Y:$FFC0,11,1 ; Start (Run) Input
#define INP_05_M M24
#define INP_05_ADR Y:$FFC0,12,1 ; Step/Quit Input
#define INP_06_M M25
#define INP_06_ADR Y:$FFC0,13,1 ; Stop (Abort) Input
#define INP_07_M M26
#define INP_07_ADR Y:$FFC0,14,1 ; Home Command Input
#define INP_08_M M27
#define INP_08_ADR Y:$FFC0,15,1 ; Feed Hold Input
#define INP_09_M M28
#define INP_09_ADR Y:$FFC0,16,1 ; Motor/C.S. Select Bit 0
#define INP_10_M M29
#define INP_10_ADR Y:$FFC0,17,1 ; Motor/C.S. Select Bit 1
#define INP_11_M M30
#define INP_11_ADR Y:$FFC0,18,1 ; Motor/C.S. Select Bit 2
#define INP_12_M M31
#define INP_12_ADR Y:$FFC0,19,1 ; Motor/C.S. Select Bit 3
#define INP_WORD_P P1 ; Storage variable for inputs
#define INP_CHANGE_P P2 ; Memory of inputs at last scan
#define INP_LATCH_P P3 ; CMD latch to pick ELSE to perform

Define M-Variable Definitions

INP_01_M->INP_01_ADR
INP_02_M->INP_02_ADR
INP_03_M->INP_03_ADR
INP_04_M->INP_04_ADR
INP_05_M->INP_05_ADR
INP_06_M->INP_06_ADR
INP_07_M->INP_07_ADR
INP_08_M->INP_08_ADR
INP_09_M->INP_09_ADR
INP_10_M->INP_10_ADR
INP_11_M->INP_11_ADR
INP_12_M->INP_12_ADR

I2 = 1 ; deactivate actual panel when running this PLC

OPEN PLC 1 CLEAR

//first combine all inputs into one word. This could be done easier for
// that are on the same connector but this shows the concept of
// building a // word from bits.

INP_WORD_P = INP_01_M + INP_02_M*2 + INP_03_M*4+ INP_04_M*8+
INP_05_M*16+ INP_06_M*32+ INP_07_M*64+ INP_08_M*128+ INP_09_M*256+
INP_10_M*512+ INP_11_M*1024+ INP_12_M*2048

IF (INP_CHANGE_P != INP_WORD_P) // has anything changed
INP_CHANGE_P = INP_WORD_P // remember last state

//put 9 to 12 together and first because they control the
//addressed motor/CS for all CMDS

IF (INP_WORD_P & $F00 =$E00)
ADDRESS #1&1
ELSE IF (INP_WORD_P & $F00 =$D00)
ADDRESS #2&2
ELSE IF (INP_WORD_P & $F00 =$C00)
ADDRESS #3&3
ELSE IF (INP_WORD_P & $F00 =$B00)
ADDRESS #4&4
ELSE IF (INP_WORD_P & $F00 =$A00)
ADDRESS #5&5
ELSE IF (INP_WORD_P & $F00 =$900)
ADDRESS #6&6
ELSE IF (INP_WORD_P & $F00 =$800)
ADDRESS #7&7
ELSE IF (INP_WORD_P & $F00 =$700)
ADDRESS #8&8
ENDIF ENDIF ENDIF ENDIF ENDIF ENDIF ENDIF ENDIF

//check for input 1
IF (INP_WORD_P & 1 != 1 AND INP_LATCH_P & 1 != 1)
INP_LATCH_P = INP_LATCH_P | 1
CMD"J-"
ELSE IF (INP_WORD_P & 1 = 1 AND INP_LATCH_P & 1 = 1)
INP_LATCH_P = INP_LATCH_P & (INP_LATCH_P^1)
CMD"J/
ENDIF ENDIF
//check for input 2
IF (INP_WORD_P & 2 != 2 AND INP_LATCH_P & 2 != 2)
INP_LATCH_P = INP_LATCH_P | 2
CMD"J+
ELSE IF (INP_WORD_P & 2 = 2 AND INP_LATCH_P & 2 = 2)
INP_LATCH_P = INP_LATCH_P & (INP_LATCH_P^2)
CMD"J/
ENDIF ENDIF

//check for input 3
IF (INP_WORD_P & 4 != 4 AND INP_LATCH_P & 4 != 4)
INP_LATCH_P = INP_LATCH_P | 4
CMD"J="
ELSE IF (INP_WORD_P & 4 = 4 AND INP_LATCH_P & 4 = 4)
INP_LATCH_P = INP_LATCH_P & (INP_LATCH_P^4)
ENDIF ENDIF

//check for input 4
IF (INP_WORD_P & 8 != 8 AND INP_LATCH_P & 8 != 8)
INP_LATCH_P = INP_LATCH_P | 8
CMD"R"
ELSE IF (INP_WORD_P & 8 = 8 AND INP_LATCH_P & 8 = 8)
INP_LATCH_P = INP_LATCH_P & (INP_LATCH_P^8)
ENDIF ENDIF

//check for input 5
IF (INP_WORD_P & 16 != 16 AND INP_LATCH_P & 16 != 16)
INP_LATCH_P = INP_LATCH_P | 16
CMD"S"
ELSE IF (INP_WORD_P & 16 = 16 AND INP_LATCH_P & = 16)
INP_LATCH_P = INP_LATCH_P & (INP_LATCH_P^16)
ENDIF ENDIF

//check for input 6
IF (INP_WORD_P & 32 != 32 AND INP_LATCH_P & 32 != 32)
INP_LATCH_P = INP_LATCH_P | 32
CMD"A"
ELSE IF (INP_WORD_P & 32 = 32 AND INP_LATCH_P & 32 = 32)
INP_LATCH_P = INP_LATCH_P & (INP_LATCH_P^32)
ENDIF ENDIF

//check for input 7
IF (INP_WORD_P & 64 != 64 AND INP_LATCH_P & 64 != 64)
INP_LATCH_P = INP_LATCH_P | 64
CMD"HM"
ELSE IF (INP_WORD_P & 64 = 64 AND INP_LATCH_P & 64 = 64)
INP_LATCH_P = INP_LATCH_P & (INP_LATCH_P^64)
ENDIF ENDIF

//check for input 8
IF (INP_WORD_P & 128 != 128 AND INP_LATCH_P & 128 != 128)
INP_LATCH_P = INP_LATCH_P | 128
CMD"H"
ELSE IF (INP_WORD_P & 128 = 128 AND INP_LATCH_P & 128 = 128)
INP_LATCH_P = INP_LATCH_P & (INP_LATCH_P^128)
ENDIF ENDIF
ENDIF CLOSE
**General I/O PLC Structure**

A lot of the work involved in writing a PLC program is that of organizing and designing the general structure. The following code can be used as a skeleton for an I/O PLC. It uses the One Time Read/Write method, latching, and MACRO definitions. The structure for triggering an action whenever an input changes state is already written. The actual actions only need to be added.

This PLC uses some of the new functions. These are the @SET_ON (variable, bit mask), @SET_OFF (variable, bit mask), @ON (variable, bit mask), @OFF (variable, bit mask). These are functions built into the PMAC Executive program’s downloader to make it easier to use bit-masking in the programs. The downloader substitutes the following PMAC code when it sees one of these functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@SET_ON(variable, bit mask)</td>
<td>variable = variable</td>
</tr>
<tr>
<td>@SET_OFF(variable, bit mask)</td>
<td>variable = variable &amp; (variable ^ bit mask)</td>
</tr>
<tr>
<td>@ON(variable, bit mask)</td>
<td>variable &amp; bit mask = bit mask</td>
</tr>
<tr>
<td>@OFF(variable, bit mask)</td>
<td>variable &amp; bit mask != bit mask</td>
</tr>
</tbody>
</table>

The rest of the programming techniques in this program have been covered earlier. This is the PLC:

```plaintext
CLOSE
ENDG
DEL GAT

#DEFINE INP_ALL_M M50
#DEFINE INP_ALL_ADR
#DEFINE MEM_INF_ALL_M M51
#DEFINE MEM_INF_ALL_ADR D:$770
#DEFINE INF_LATCH_M M52
#DEFINE INF_LATCH_ADR D:$771
INF_ALL_M->INF_ALL_ADR
MEM_INF_ALL_M->MEM_INF_ALL_ADR
INF_LATCH_M->INF_LATCH_ADR

#DEFINE MEM_MASK 01 $1
#DEFINE MEM_MASK_02 $2
#DEFINE MEM_MASK_03 $4
#DEFINE MEM_MASK_04 $8
#DEFINE MEM_MASK_05 $10
#DEFINE MEM_MASK_06 $20
#DEFINE MEM_MASK_07 $40
#DEFINE MEM_MASK_08 $80
#DEFINE MEM_MASK_09 $100
#DEFINE MEM_MASK_10 $200
#DEFINE MEM_MASK_11 $400
#DEFINE MEM_MASK_12 $800
#DEFINE MEM_MASK_13 $1000
#DEFINE MEM_MASK_14 $2000
#DEFINE MEM_MASK_15 $4000
#DEFINE MEM_MASK_16 $8000
#DEFINE MEM_MASK_17 $10000
#DEFINE MEM_MASK_18 $20000
#DEFINE MEM_MASK_19 $40000
#DEFINE MEM_MASK_20 $80000
#DEFINE MEM_MASK_21 $100000
#DEFINE MEM_MASK_22 $200000
#DEFINE MEM_MASK_23 $400000
#DEFINE MEM_MASK_24 $800000
#DEFINE MEM_MASK_25 $1000000
#DEFINE MEM_MASK_26 $2000000
#DEFINE MEM_MASK_27 $4000000
#DEFINE MEM_MASK_28 $8000000
#DEFINE MEM_MASK_29 $10000000
#DEFINE MEM_MASK_30 $20000000
```
OPEN PLC 1 CLEAR
IF(INP_ALL_M ! = MEM_INP_ALL_M)
   MEM_INP_ALL_M = INF_ALL_M
ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_01) AND @OFF(INP_LATCH_M, MEM_MASK_01))
   @SET_ON(INP_LATCH_M, MEM_MASK_01))
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_01) AND @ON(INP_LATCH_M, MEM_MASK_01))
   @SET_OFF(INP_LATCH_M, MEM_MASK_01))
ENDIF
ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_02) AND @OFF(INP_LATCH_M, MEM_MASK_02))
   @SET_ON(INP_LATCH_M, MEM_MASK_02))
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_02) AND @ON(INP_LATCH_M, MEM_MASK_02))
   @SET_OFF(INP_LATCH_M, MEM_MASK_02))
ENDIF
ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_03) AND @OFF(INP_LATCH_M, MEM_MASK_03))
   @SET_ON(INP_LATCH_M, MEM_MASK_03))
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_03) AND @ON(INP_LATCH_M, MEM_MASK_03))
   @SET_OFF(INP_LATCH_M, MEM_MASK_03))
ENDIF
ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_04) AND @OFF(INP_LATCH_M, MEM_MASK_04))
   @SET_ON(INP_LATCH_M, MEM_MASK_04))
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_04) AND @ON(INP_LATCH_M, MEM_MASK_04))
   @SET_OFF(INP_LATCH_M, MEM_MASK_04))
ENDIF
ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_05) AND @OFF(INP_LATCH_M, MEM_MASK_05))
   @SET_ON(INP_LATCH_M, MEM_MASK_05))
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_05) AND @ON(INP_LATCH_M, MEM_MASK_05))
   @SET_OFF(INP_LATCH_M, MEM_MASK_05))
ENDIF
ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_06) AND @OFF(INP_LATCH_M, MEM_MASK_06))
   @SET_ON(INP_LATCH_M, MEM_MASK_06))
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_06) AND @ON(INP_LATCH_M, MEM_MASK_06))
   @SET_OFF(INP_LATCH_M, MEM_MASK_06))
ENDIF
ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_07) AND @OFF(INP_LATCH_M, MEM_MASK_07))
   @SET_ON(INP_LATCH_M, MEM_MASK_07))
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_07) AND @ON(INP_LATCH_M, MEM_MASK_07))
   @SET_OFF(INP_LATCH_M, MEM_MASK_07))
ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_08) AND @OFF(INP_LATCH_M, MEM_MASK_08))
@SET_ON(INP_LATCH_M, MEM_MASK_08)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_08) AND @ON(INP_LATCH_M, MEM_MASK_08))
@SET_OFF(INP_LATCH_M, MEM_MASK_08)
ENDIF
ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_09) AND @OFF(INP_LATCH_M, MEM_MASK_09))
@SET_ON(INP_LATCH_M, MEM_MASK_09)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_09) AND @ON(INP_LATCH_M, MEM_MASK_09))
@SET_OFF(INP_LATCH_M, MEM_MASK_09)
ENDIF
ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_10) AND @OFF(INP_LATCH_M, MEM_MASK_10))
@SET_ON(INP_LATCH_M, MEM_MASK_10)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_10) AND @ON(INP_LATCH_M, MEM_MASK_10))
@SET_OFF(INP_LATCH_M, MEM_MASK_10)
ENDIF
ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_11) AND @OFF(INP_LATCH_M, MEM_MASK_11))
@SET_ON(INP_LATCH_M, MEM_MASK_11)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_11) AND @ON(INP_LATCH_M, MEM_MASK_11))
@SET_OFF(INP_LATCH_M, MEM_MASK_11)
ENDIF
ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_12) AND @OFF(INP_LATCH_M, MEM_MASK_12))
@SET_ON(INP_LATCH_M, MEM_MASK_12)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_12) AND @ON(INP_LATCH_M, MEM_MASK_12))
@SET_OFF(INP_LATCH_M, MEM_MASK_12)
ENDIF
ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_13) AND @OFF(INP_LATCH_M, MEM_MASK_13))
@SET_ON(INP_LATCH_M, MEM_MASK_13)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_13) AND @ON(INP_LATCH_M, MEM_MASK_13))
@SET_OFF(INP_LATCH_M, MEM_MASK_13)
ENDIF
ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_14) AND @OFF(INP_LATCH_M, MEM_MASK_14))
@SET_ON(INP_LATCH_M, MEM_MASK_14)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_14) AND @ON(INP_LATCH_M, MEM_MASK_14))
@SET_OFF(INP_LATCH_M, MEM_MASK_14)
ENDIF
ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_15) AND @OFF(INP_LATCH_M, MEM_MASK_15))
@SET_ON(INP_LATCH_M, MEM_MASK_15)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_15) AND @ON(INP_LATCH_M, MEM_MASK_15))
@SET_OFF(INP_LATCH_M, MEM_MASK_15)
ENDIF
ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_16) AND @OFF(INP_LATCH_M, MEM_MASK_16))
@SET_ON(INP_LATCH_M, MEM_MASK_16)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_16) AND @ON(INP_LATCH_M, MEM_MASK_16))
@SET_OFF(INP_LATCH_M, MEM_MASK_16)
ENDIF
ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_17) AND @OFF(INP_LATCH_M, MEM_MASK_17))
@SET_ON(INP_LATCH_M, MEM_MASK_17)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_17) AND @ON(INP_LATCH_M, MEM_MASK_17))
@SET_OFF(INP_LATCH_M, MEM_MASK_17)
ENDIF
ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_18) AND @OFF(INP_LATCH_M, MEM_MASK_18))
@SET_ON(INP_LATCH_M, MEM_MASK_18)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_18) AND @ON(INP_LATCH_M, MEM_MASK_18))
@SET_OFF(INP_LATCH_M, MEM_MASK_18)
ENDIF
ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_19) AND @OFF(INP_LATCH_M, MEM_MASK_19))
  @SET_ON(INP_LATCH_M, MEM_MASK_19)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_19) AND @ON(INP_LATCH_M, MEM_MASK_19))
  @SET_OFF(INP_LATCH_M, MEM_MASK_19)
ENDIF ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_20) AND @OFF(INP_LATCH_M, MEM_MASK_20))
  @SET_ON(INP_LATCH_M, MEM_MASK_20)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_20) AND @ON(INP_LATCH_M, MEM_MASK_20))
  @SET_OFF(INP_LATCH_M, MEM_MASK_20)
ENDIF ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_21) AND @OFF(INP_LATCH_M, MEM_MASK_21))
  @SET_ON(INP_LATCH_M, MEM_MASK_21)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_21) AND @ON(INP_LATCH_M, MEM_MASK_21))
  @SET_OFF(INP_LATCH_M, MEM_MASK_21)
ENDIF ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_22) AND @OFF(INP_LATCH_M, MEM_MASK_22))
  @SET_ON(INP_LATCH_M, MEM_MASK_22)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_22) AND @ON(INP_LATCH_M, MEM_MASK_22))
  @SET_OFF(INP_LATCH_M, MEM_MASK_22)
ENDIF ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_23) AND @OFF(INP_LATCH_M, MEM_MASK_23))
  @SET_ON(INP_LATCH_M, MEM_MASK_23)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_23) AND @ON(INP_LATCH_M, MEM_MASK_23))
  @SET_OFF(INP_LATCH_M, MEM_MASK_23)
ENDIF ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_24) AND @OFF(INP_LATCH_M, MEM_MASK_24))
  @SET_ON(INP_LATCH_M, MEM_MASK_24)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_24) AND @ON(INP_LATCH_M, MEM_MASK_24))
  @SET_OFF(INP_LATCH_M, MEM_MASK_24)
ENDIF ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_25) AND @OFF(INP_LATCH_M, MEM_MASK_25))
  @SET_ON(INP_LATCH_M, MEM_MASK_25)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_25) AND @ON(INP_LATCH_M, MEM_MASK_25))
  @SET_OFF(INP_LATCH_M, MEM_MASK_25)
ENDIF ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_26) AND @OFF(INP_LATCH_M, MEM_MASK_26))
  @SET_ON(INP_LATCH_M, MEM_MASK_26)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_26) AND @ON(INP_LATCH_M, MEM_MASK_26))
  @SET_OFF(INP_LATCH_M, MEM_MASK_26)
ENDIF ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_27) AND @OFF(INP_LATCH_M, MEM_MASK_27))
  @SET_ON(INP_LATCH_M, MEM_MASK_27)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_27) AND @ON(INP_LATCH_M, MEM_MASK_27))
  @SET_OFF(INP_LATCH_M, MEM_MASK_27)
ENDIF ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_28) AND @OFF(INP_LATCH_M, MEM_MASK_28))
  @SET_ON(INP_LATCH_M, MEM_MASK_28)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_28) AND @ON(INP_LATCH_M, MEM_MASK_28))
  @SET_OFF(INP_LATCH_M, MEM_MASK_28)
ENDIF ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_29) AND @OFF(INP_LATCH_M, MEM_MASK_29))
  @SET_ON(INP_LATCH_M, MEM_MASK_29)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_29) AND @ON(INP_LATCH_M, MEM_MASK_29))
  @SET_OFF(INP_LATCH_M, MEM_MASK_29)
ENDIF ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_30) AND @OFF(INP_LATCH_M, MEM_MASK_30))
  @SET_ON(INP_LATCH_M, MEM_MASK_30)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_30) AND @ON(INP_LATCH_M, MEM_MASK_30))
  @SET_OFF(INP_LATCH_M, MEM_MASK_30)
ENDIF ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_31) AND @OFF(INP_LATCH_M, MEM_MASK_31))
  @SET_ON(INP_LATCH_M, MEM_MASK_31)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_31) AND @ON(INP_LATCH_M, MEM_MASK_31))
  @SET_OFF(INP_LATCH_M, MEM_MASK_31)
ENDIF ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_32) AND @OFF(INP_LATCH_M, MEM_MASK_32))
  @SET_ON(INP_LATCH_M, MEM_MASK_32)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_32) AND @ON(INP_LATCH_M, MEM_MASK_32))
  @SET_OFF(INP_LATCH_M, MEM_MASK_32)
ENDIF ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_33) AND @OFF(INP_LATCH_M, MEM_MASK_33))
  @SET_ON(INP_LATCH_M, MEM_MASK_33)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_33) AND @ON(INP_LATCH_M, MEM_MASK_33))
  @SET_OFF(INP_LATCH_M, MEM_MASK_33)
ENDIF ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_34) AND @OFF(INP_LATCH_M, MEM_MASK_34))
  @SET_ON(INP_LATCH_M, MEM_MASK_34)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_34) AND @ON(INP_LATCH_M, MEM_MASK_34))
  @SET_OFF(INP_LATCH_M, MEM_MASK_34)
ENDIF ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_35) AND @OFF(INP_LATCH_M, MEM_MASK_35))
  @SET_ON(INP_LATCH_M, MEM_MASK_35)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_35) AND @ON(INP_LATCH_M, MEM_MASK_35))
  @SET_OFF(INP_LATCH_M, MEM_MASK_35)
ENDIF ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_36) AND @OFF(INP_LATCH_M, MEM_MASK_36))
  @SET_ON(INP_LATCH_M, MEM_MASK_36)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_36) AND @ON(INP_LATCH_M, MEM_MASK_36))
  @SET_OFF(INP_LATCH_M, MEM_MASK_36)
ENDIF ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_37) AND @OFF(INP_LATCH_M, MEM_MASK_37))
  @SET_ON(INP_LATCH_M, MEM_MASK_37)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_37) AND @ON(INP_LATCH_M, MEM_MASK_37))
  @SET_OFF(INP_LATCH_M, MEM_MASK_37)
ENDIF ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_38) AND @OFF(INP_LATCH_M, MEM_MASK_38))
  @SET_ON(INP_LATCH_M, MEM_MASK_38)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_38) AND @ON(INP_LATCH_M, MEM_MASK_38))
  @SET_OFF(INP_LATCH_M, MEM_MASK_38)
ENDIF ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_39) AND @OFF(INP_LATCH_M, MEM_MASK_39))
  @SET_ON(INP_LATCH_M, MEM_MASK_39)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_39) AND @ON(INP_LATCH_M, MEM_MASK_39))
  @SET_OFF(INP_LATCH_M, MEM_MASK_39)
ENDIF ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_40) AND @OFF(INP_LATCH_M, MEM_MASK_40))
  @SET_ON(INP_LATCH_M, MEM_MASK_40)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_40) AND @ON(INP_LATCH_M, MEM_MASK_40))
  @SET_OFF(INP_LATCH_M, MEM_MASK_40)
ENDIF ENDIF
IF((@ON(MEM_INP_ALL, MEM_MASK_41) AND @OFF(INP_LATCH_M, MEM_MASK_41))
    @SET_ON(INP_LATCH_M, MEM_MASK_41)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_41) AND @ON(INP_LATCH_M, MEM_MASK_41))
    @SET_OFF(INP_LATCH_M, MEM_MASK_41)
ENDIF ENDIF

IF((@ON(MEM_INP_ALL, MEM_MASK_42) AND @OFF(INP_LATCH_M, MEM_MASK_42))
    @SET_ON(INP_LATCH_M, MEM_MASK_42)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_42) AND @ON(INP_LATCH_M, MEM_MASK_42))
    @SET_OFF(INP_LATCH_M, MEM_MASK_42)
ENDIF ENDIF

IF((@ON(MEM_INP_ALL, MEM_MASK_43) AND @OFF(INP_LATCH_M, MEM_MASK_43))
    @SET_ON(INP_LATCH_M, MEM_MASK_43)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_43) AND @ON(INP_LATCH_M, MEM_MASK_43))
    @SET_OFF(INP_LATCH_M, MEM_MASK_43)
ENDIF ENDIF

IF((@ON(MEM_INP_ALL, MEM_MASK_44) AND @OFF(INP_LATCH_M, MEM_MASK_44))
    @SET_ON(INP_LATCH_M, MEM_MASK_44)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_44) AND @ON(INP_LATCH_M, MEM_MASK_44))
    @SET_OFF(INP_LATCH_M, MEM_MASK_44)
ENDIF ENDIF

IF((@ON(MEM_INP_ALL, MEM_MASK_45) AND @OFF(INP_LATCH_M, MEM_MASK_45))
    @SET_ON(INP_LATCH_M, MEM_MASK_45)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_45) AND @ON(INP_LATCH_M, MEM_MASK_45))
    @SET_OFF(INP_LATCH_M, MEM_MASK_45)
ENDIF ENDIF

IF((@ON(MEM_INP_ALL, MEM_MASK_46) AND @OFF(INP_LATCH_M, MEM_MASK_46))
    @SET_ON(INP_LATCH_M, MEM_MASK_46)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_46) AND @ON(INP_LATCH_M, MEM_MASK_46))
    @SET_OFF(INP_LATCH_M, MEM_MASK_46)
ENDIF ENDIF

IF((@ON(MEM_INP_ALL, MEM_MASK_47) AND @OFF(INP_LATCH_M, MEM_MASK_47))
    @SET_ON(INP_LATCH_M, MEM_MASK_47)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_47) AND @ON(INP_LATCH_M, MEM_MASK_47))
    @SET_OFF(INP_LATCH_M, MEM_MASK_47)
ENDIF ENDIF

IF((@ON(MEM_INP_ALL, MEM_MASK_48) AND @OFF(INP_LATCH_M, MEM_MASK_48))
    @SET_ON(INP_LATCH_M, MEM_MASK_48)
ELSE IF((@OFF(MEM_INP_ALL, MEM_MASK_48) AND @ON(INP_LATCH_M, MEM_MASK_48))
    @SET_OFF(INP_LATCH_M, MEM_MASK_48)
ENDIF ENDIF

ENDIF

CLOSE