SIMATIC
S7-200 Tips

Group	Topic
4	**Freeport Communication Interface to SIMOVERT Motor Drive**

Overview

In this example, the CPU 214 communicates with a SIMOVERT MicroMaster motor drive to start, stop, and vary the output frequency to the motor. Communication occurs in the S7-200 Freeport mode using the USS 5-Word Protocol. An input simulator is used to initiate commands to the motor drive.

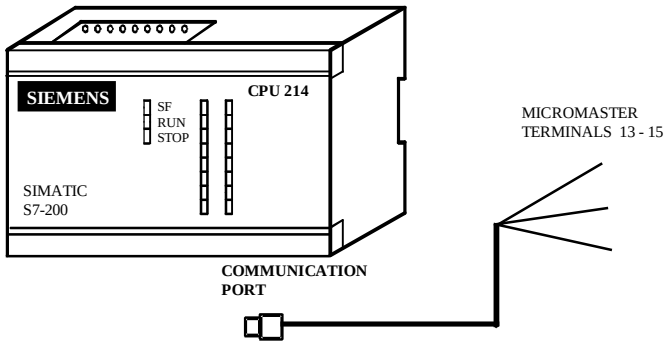


Figure 28.1

Hardware Requirements

This program assumes that the user has appropriately wired a motor and the MicroMaster motor drive, and that all motor and MicroMaster parameters have been set up manually. The MicroMaster must also be set to operate in remote mode (P910 = 1) with its baud rate set to 19.2Kb (P92 = 7) and MicroMaster address set to 1 (P91 = 1).

A communication cable (with a 9-pin male connector to the CPU 214, having pins 1, 3 and 8 connected) is required. The other end of the cable should be open-ended so that the 3 wires can be connected to terminals 13 - 15 on the MicroMaster (pin 1 to terminal 15, pin 3 to terminal 13, pin 8 to terminal 14).

Program Structure

MAIN	*The main program* monitors /TERM/STOP switch for Freeport/PPI communication & looks for input point rising edges as motor commands.
SBR 0	*Setup for Freeport communication* sets up Freeport mode parameters on first scan.
SBR 1	*RUN subroutine* sets up motor to run at constant speed.
SBR 2	*RAMP subroutine* sets up motor to run at variable speed.
SBR 3	*INC frequency multiplier subroutine* increases MicroMaster output frequency.
SBR 4	*DEC frequency multiplier subroutine* decreases MicroMaster output frequency.
SBR 5	*STOP subroutine* stops the motor.
SBR 6	*Outgoing message BCC calculation*.
SBR 7	*Transmit message & initiate transmit timeout timing*.
INT 0	*Transmit complete interrupt handler* turns on the receiver.
INT 1	*Transmit timeout interrupt handler* retries transmit for up to 3 timeouts.
INT 2	*Receive characters interrupt routine* performs validity checks after completion.
INT 3	*Receive timeout interrupt handler* retries transmit for up to 3 timeouts.

Program Description

The CPU 214 communicates with a SIMOVERT MicroMaster motor drive to start, stop, and vary the output frequency to the motor. Communication is via the S7-200 Freeport mode using the USS 5-Word Protocol. An input simulator is used to initiate commands to the motor drive. The program monitors the RUN/TERM switch and sets the Freeport mode control byte(SMB30) protocol selection field accordingly. It also monitors input points for motor commands:

I0.0 rising edge	Run motor at constant frequency as of last command.
I0.2 rising edge	Run motor at variable frequency starting at frequency of last command. Frequency can be increased/decreased using I0.6 & I0.7.
I0.4 rising edge	Stop the motor.
I0.5 level	1x or 2x multiplier for amount of change in frequency. I0.5: 0 1x, 1 2x.
I0.6 rising edge	Increase frequency of motor by 1x or 2x the frequency delta(50 in this example).
I0.7 rising edge	Decrease frequency of motor by 1x or 2x the frequency delta(50 in this example).
I1.0 level	Motor direction. I1.0: 0 CW, 1 CCW

The program detects and reports communication errors. Transmission to the MicroMaster is timed, and it is attempted up to 3 times before aborting the specified operation. The subsequent receive operation from the MicroMaster is timed, and the transmit/receive operation is also attempted up to 3 times before aborting the specified operation. The response message from the MicroMaster is checked for validity (STX, LEN, ADR and BCC). Any error detected is shown in QB0 until completion of the next operation:

0	No error.
1	Illegal response(other than bad BCC).
2	Bad BCC.
3	Transmit timeout.
4	Receive timeout.

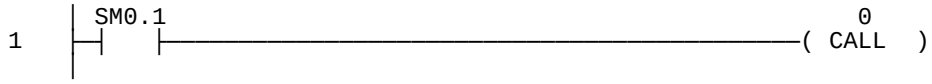
Although this sample program communicates with only one MicroMaster, it could be extended to use additional input points to select the address of a particular MicroMaster on a multidrop link to which a command is sent. Additionally, the basic communication structure of this program may be used to send other messages to a MicroMaster, i.e., monitoring current, torque, etc.

The program size is 342 words.

Main Program

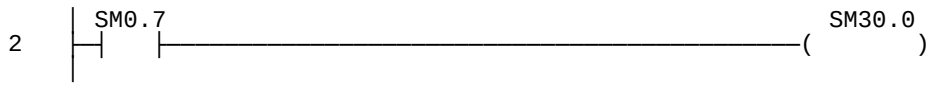
```
// This program is an example of S7-200 Freeport communication interfacing with
// a SIMOVERT MicroMaster motor drive.  The communication between the two
// uses the USS 5-Word Protocol.  The S7-200 is acting as the master and, in this
// example, is acting as USS Protocol address 0, while the MicroMaster is at address 1.

// For the described application you need:
//        1 SIMATIC S7-200 CPU 212 or CPU 214 with similator for inputs.
//        1 RS 485 cable with one male interface and an open end to connect to the
//        MicroMaster.
//        1 MicroMaster Drive & motor.
//        ***This example used a demo unit(drive & motor) from Herb Taylor, SE&A.***
//
// Ensure that the MicroMaster parameters are set appropriately for the motor you
// are using.  Also ensure that the MicroMaster is set for remote mode (P910 = 1), baud
// rate select is appropriately set (P092) and that the MicroMaster slave address is set
// (P091).  For this example, baud rate is 19.2Kb and MicroMaster address is 1.

//  The USS Serial Communications 5-Word Protocol as used by the MicroMaster
//  has the following 14-byte structure:
//
//        02              Start of message (STX).
//        12              Length of message (12 bytes)with the MicroMaster (from AA to
//                        BCC).
//        AA              Device number address (In this Demo the Drive is 1)
//        PKE             High byte used for parameterisation control.
//        PKE             Low byte
//        IND             High byte used for Array index word
//        IND             Low byte
//        PWE             High byte used for param. values and error codes.
//        PWE             Low byte
//        PZD1            High byte used for control/status word
//        PZD1            Low byte
//        PZD2            High byte used for main set-point and return values.
//        PZD2            Low byte
//        BCC             Block Checksum
//
//        ***Note that this example uses only data words
//        ***PZD1 and PZD2 for ON/OFF/Speed changes.

// Structure of the program:
//        MAIN    The main program.
//        SBR 0   Setup for Freeport communication.
//        SBR 1   RUN subroutine.
//        SBR 2   RAMP subroutine.
//        SBR 3   INC frequency multiplier subroutine.
//        SBR 4   DEC frequency multiplier subroutine.
//        SBR 5   STOP subroutine.
//        SBR 6   Outgoing message BCC calculation.
//        SBR 7   Transmit message & initiate transmit timeout timing.
//        INT 0   Transmit complete interrupt handler.  Turns on receiver.
//        INT 1   Transmit timeout interrupt handler.
//        INT 2   Receive characters interrupt routine.
//        INT 3   Receive timeout interrupt handler.
```

```
// Memory Usage:

//      VB99            Transmit msg. length
//      VB100-VB113     Transmit buffer.
//      VB114-VB127     Receive buffer.
//      VW200           Default  frequency multiplier.  Initially = 5461.
//                      Frequency = (5461/16384) x P094
//      VW202           Frequency multiplier increment/decrement amount.  Initially = 50.
//      VW204           Transmit retry count.  Set to 3 at init. & on successful XMT.
//      VW206           Receive retry count.  Set to 3 at init. & on successful resp. receive.
//      VW208           No. chars. to receive in a message.  For USS 10 = 14.
//      VB210           Status of last attempted operation (also shown in QB0):
//                              0       Operation OK.
//                              1       Illegal response from MicroMaster
//                                      (other than bad BCC).
//                              2       Bad BCC.
//                              3       Transmit timeout.
//                              4       Receive timeout.
//      VD211           Receive buffer address pointer
//      VW215           Accumulating BCC of msg. being received in lower byte.
//      VB217-VB218     Scratch area

// I/O Usage:

//      I0.0            RUN.  Turn motor ON at current value of frequency multiplier.
//                      Direction is taken from I1.0.  Affects operation only after a STOP.
//                      Rising edge operation.
//      I0.2            RAMP.  Turn motor ON at current value of frequency multiplier.
//                      Direction is taken from I1.0. Affects operation only after a STOP.
//                      Rising edge operation.
//      I0.4            STOP.  Stops motor and allows subsequent RUN/RAMP commands.
//                      Rising edge operation.
//      I0.5            Specifies multiplier(1x or 2x) for frequency multiplier increment/
//                      decrement amount.  Level operation:  0     1x, 1     2x.
//      I0.6            Increment frequency multiplier by value in VW202x multiplier (I0.5).
//                      If motor is in RAMP, speed is immediately increased.  Rising edge
//                      operation.
//      I0.7            Decrement frequency multiplier by value in VW202x multiplier (I0.5).
//                      If motor is in RAMP, speed is immediately decreased.  Rising edge
//                      operation.
//      I1.0            Specifies motor direction.  Level operation:  0     CW, 1     CCW.

// First scan maintenance activities
```



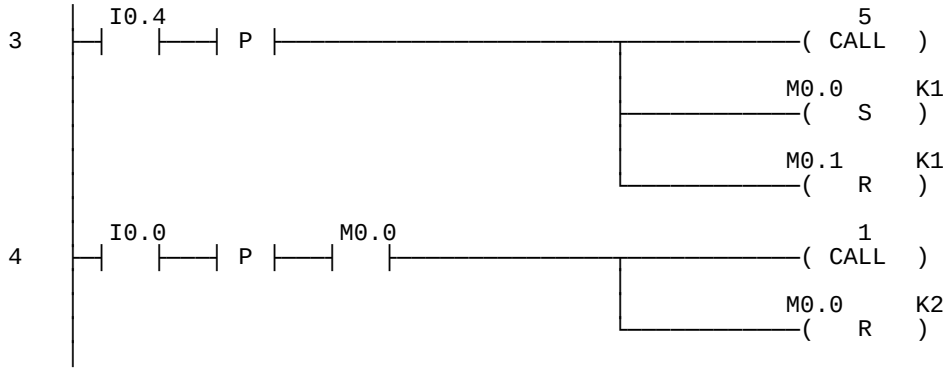

```
LD              SM0.1
CALL            0               // Set up Freeport comm. buffers, settings, etc.
```

// Follow STOP-TERM- mode switch for Freeport or PPI mode selection

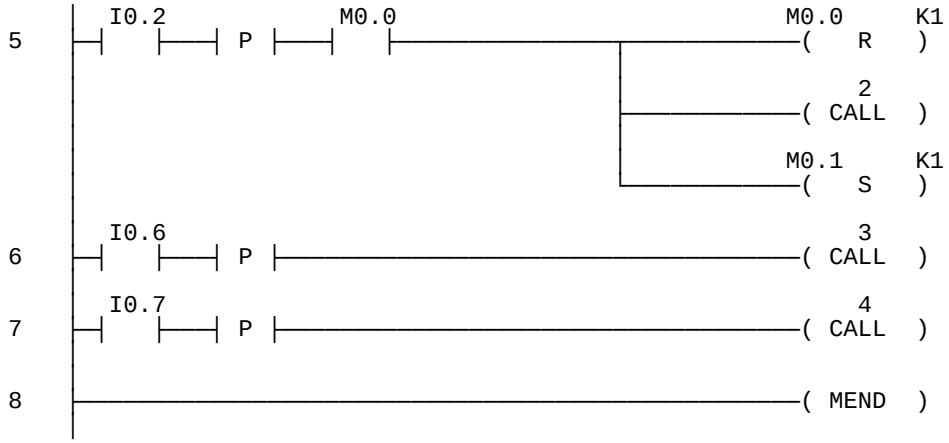



```
LD      SM0.7
=       SM30.0
```

// Check for any commands



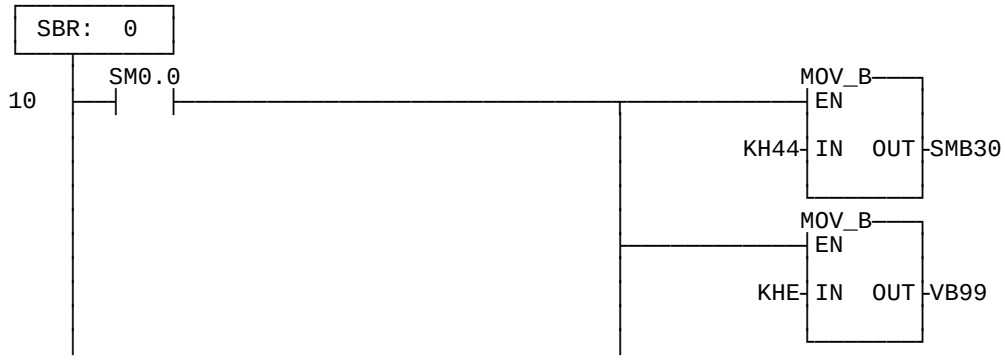

```
LD      I0.4        // STOP
EU                  // Edge?
CALL    5           // Send msg to MicroMaster
S       M0.0,1      // Enable subsequent RUN or  RAMP cmd.
R       M0.1,1      // No longer in

LD      I0.0        // RUN
EU                  // Edge?
A       M0.0        // Enabled?
CALL    1           // Send msg to MicroMaster
R       M0.0,2      // Disable subsequent RUN or  cmd. & no
```

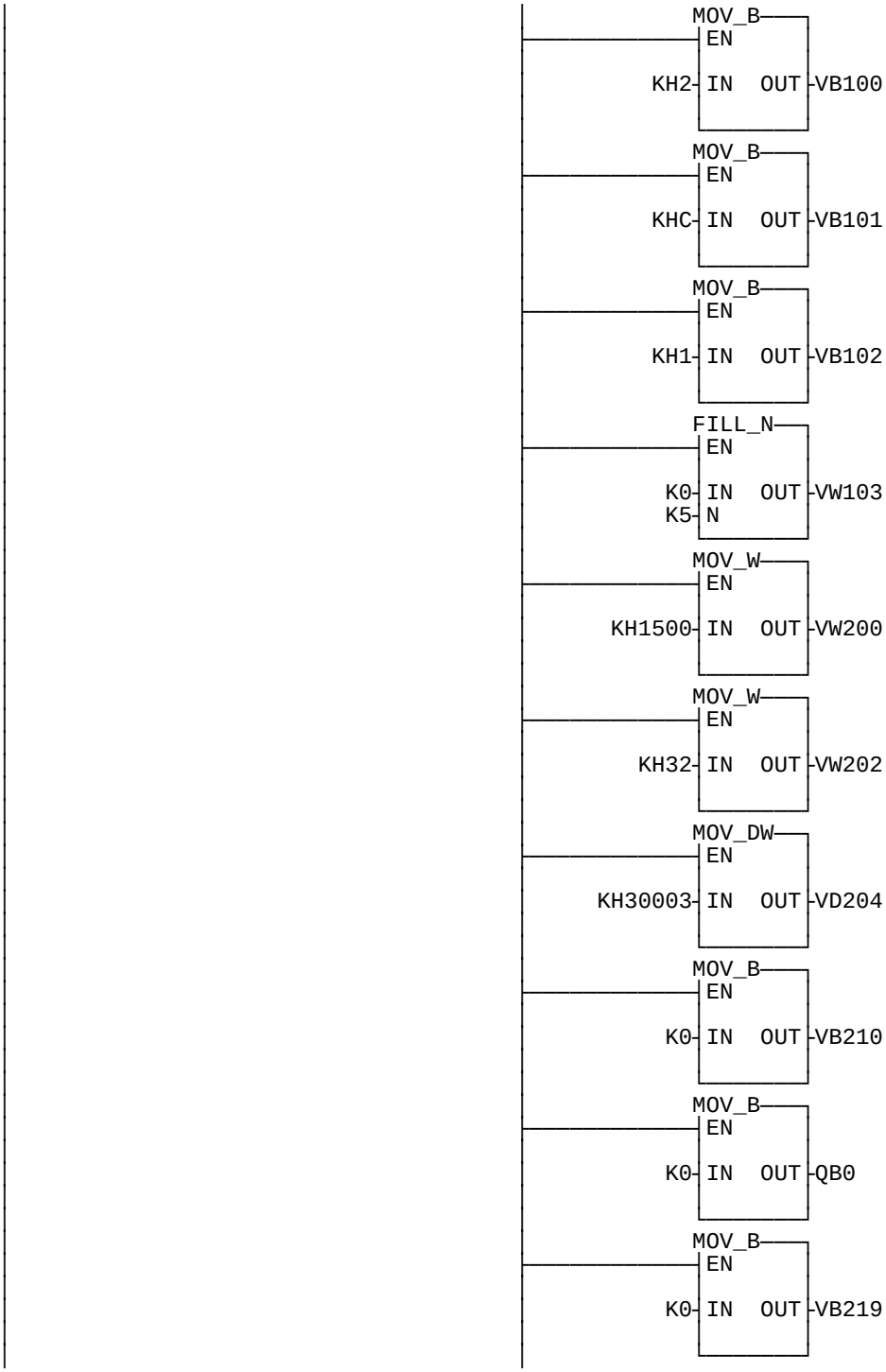



```
LD          I0.2        //
EU                      // Edge?
A           M0.0        // Enabled?
R           M0.0,1      // Disable subsequent RUN or cmd.
CALL        2           // Send msg to MicroMaster
S           M0.1,1      // Show now in state

LD          I0.6        // Increment freq.?
EU
CALL        3           // ++ speed

LD          I0.7        // Decrement freq.?
EU
CALL        4           // -- speed

MEND
```

Subroutines

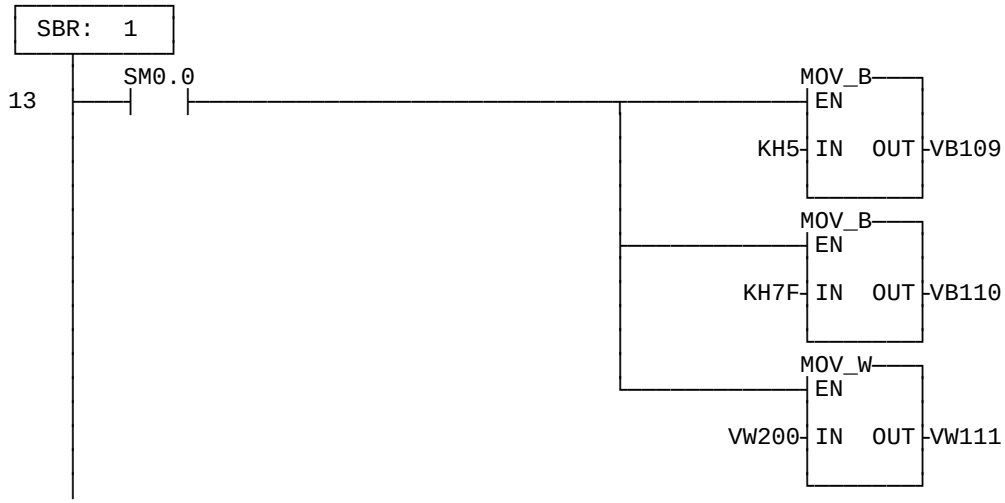
// Initialization subroutine - executed on 1st scan only



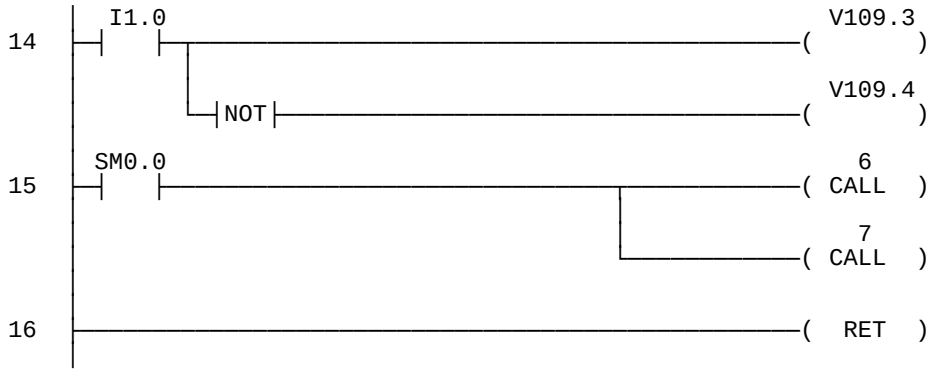



```
SBR     0              // Initializes XMT buffer and sets up comm. settings
LD      SM0.0          // Always On Bit
MOVB    16#44,SMB30    //  19.2kb, 8, Even, 1 when Freeport
MOVB    16#0E,VB99     //  XMT length
MOVB    16#02,VB100    //  STX
MOVB    16#0C,VB101    //  LEN
MOVB    16#01,VB102    //  ADR(MicroMaster is at address 1)
FILL    0,VW103,5      // Clear ALL 5 data words; can change later
MOVW    16#1500,VW200  // 1/3 max. frequency multiplier
MOVW    16#32,VW202    // Freq. multiplier inc/dec of 50
MOVD    16#00030003,VD204 // Set xmit & rcv retry counts to 3
MOVB    0,VB210// Clear operation status indicators.
MOVB    0,QB0
MOVB    0,VB219// S7-200 address
S       M0.0,1         // Allow RUN or  RAMP or RAMP( & direction change)
ENI                    // Enable user interrupts
RET
```

```
// Subroutine to handle running the motor; direction taken from I1.0.
```

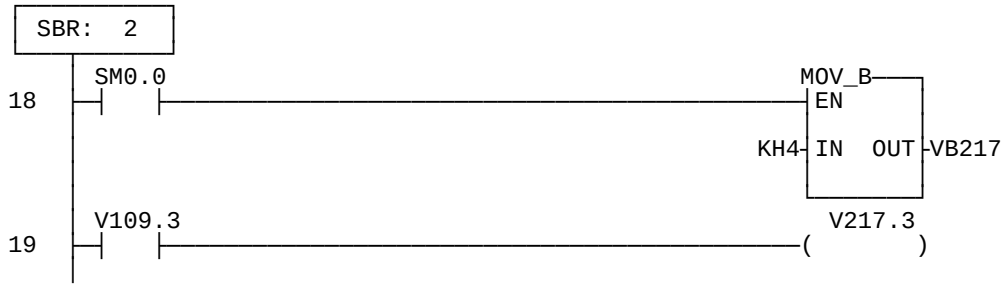



```
SBR     1              //  Run motor
LD      SM0.0          // Always On Bit
MOVB    16#05,VB109    // Set up STW for
MOVB    16#7F,VB110    // cw inching
MOVW    VW200,VW111 // Set frequency
```

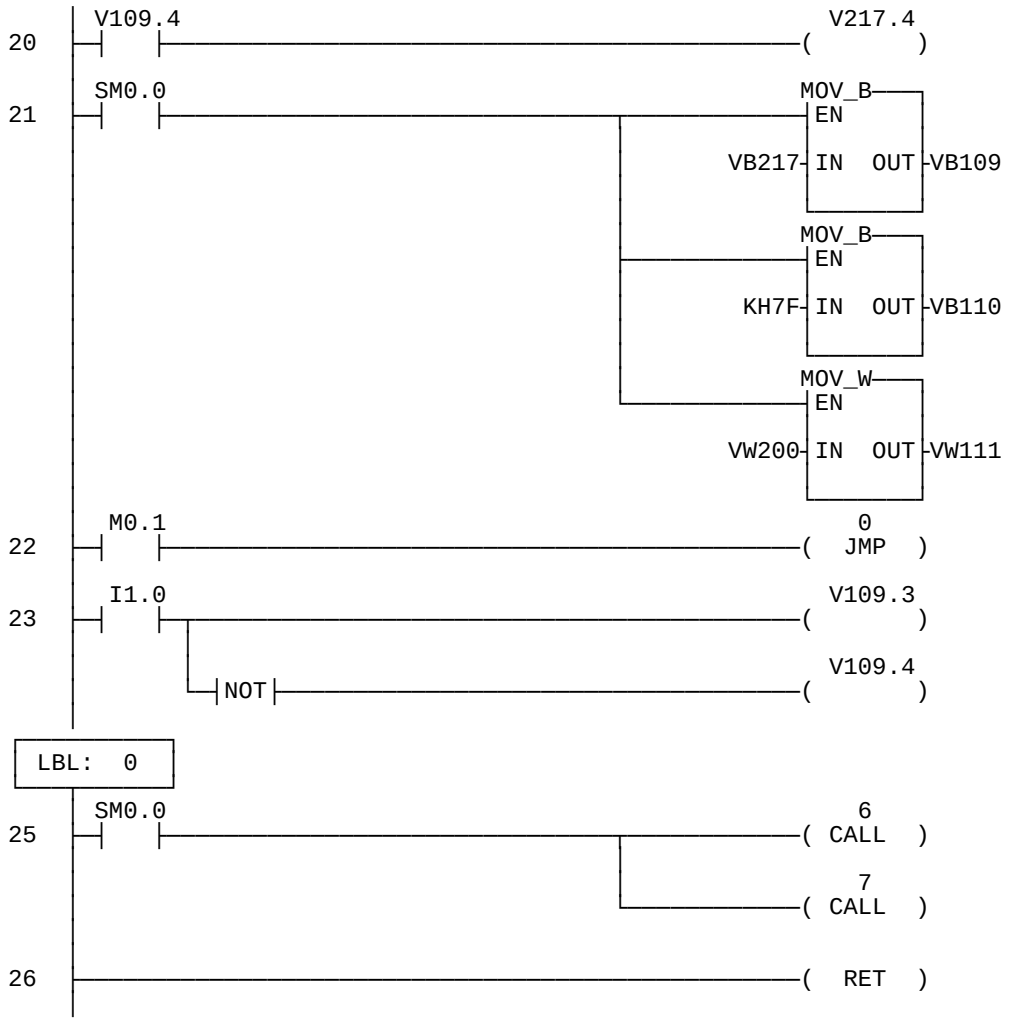



```
LD          I1.0        // Set up direction
=           V109.3      // for start
NOT
=           V109.4

LD          SM0.0
CALL        6           // Calculate BCC
CALL        7           //  Initiate/Time XMT

RET
```

```
// Subroutine to handle running/ramping the motor.  Called from seeing I0.2 edge and also
// from seeing +/- ramp modification to motor frequency when motor is running.  Direction
// taken from I1.0.
```

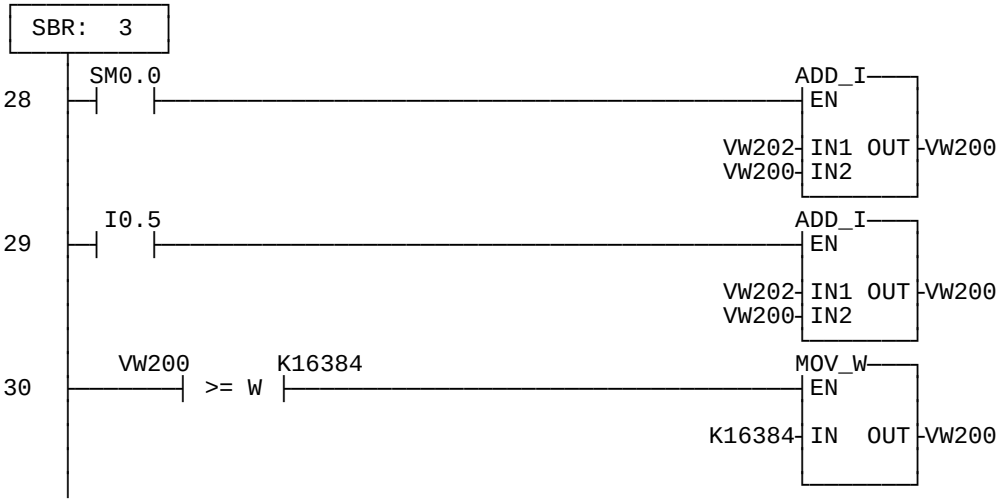



```
SBR         2           //  Run motor
LD          SM0.0       // Always On Bit
MOVB        16#04,VB217 // Set up command byte

LD          V109.3      // Preserve any
=           V217.3      // previous motor
```

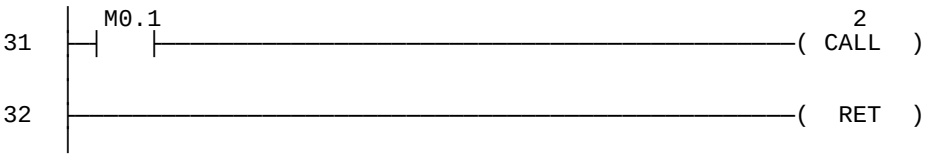



```
LD      V109.4      // direction
=       V217.4      // indicators

LD      SM0.0
MOVB    VB217,VB109 //  Set up STW for
MOVB    16#7F,VB110 //  run.
MOVW    VW200,VW111 // Set frequency

LD      M0.1        // Check state to see if direction specification allowed
JMP     0           // Only allowed on  after STOP; not in
                    // If in, previous direction is preserved
LD      I1.0        // Set up direction
=       V109.3      // for run
NOT
=       V109.4

LBL     0
LD      SM0.0
CALL    6           // Calculate BCC
CALL    7           //  Initiate/Time XMT
RET
```

```
// Subroutine to handle increasing motor frequency.  Freq. incremented by value
// in VW202.  If I0.5 is ON, it is doubly incremented.  On overflow, freq. set to
// 32767.
```

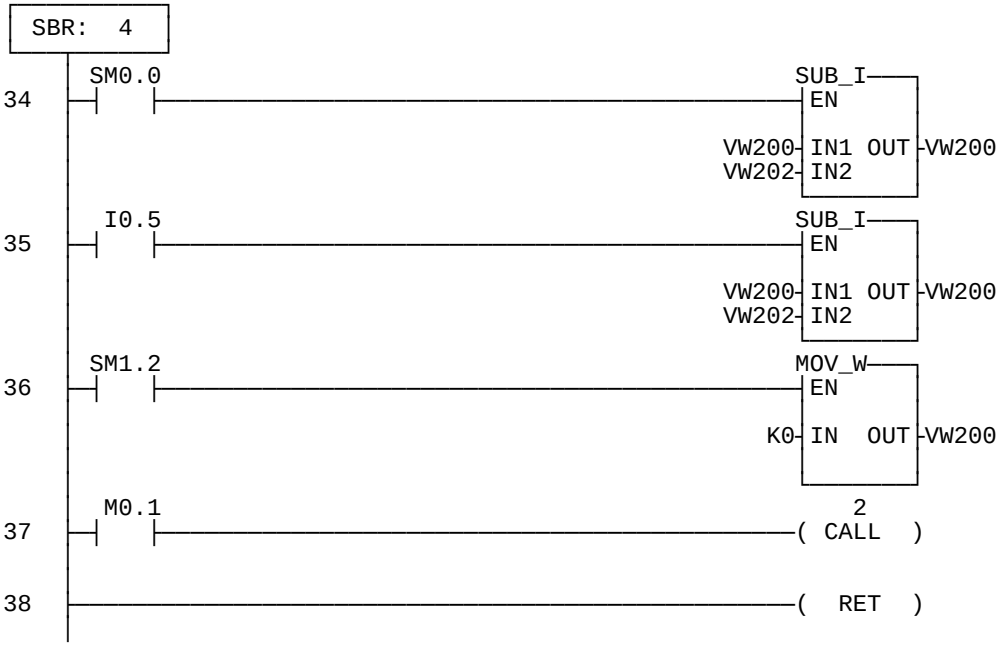



```
SBR             3               //  Increase freq.
LD              SM0.0           // Always On Bit
+I              VW202,VW200     // by +factor

LD              I0.5            // Double it?
+I              VW202,VW200

LDW>= VW200,16384    // Did we overflow max. +?
MOVW            16384,VW200     // If so, set to max. +
```



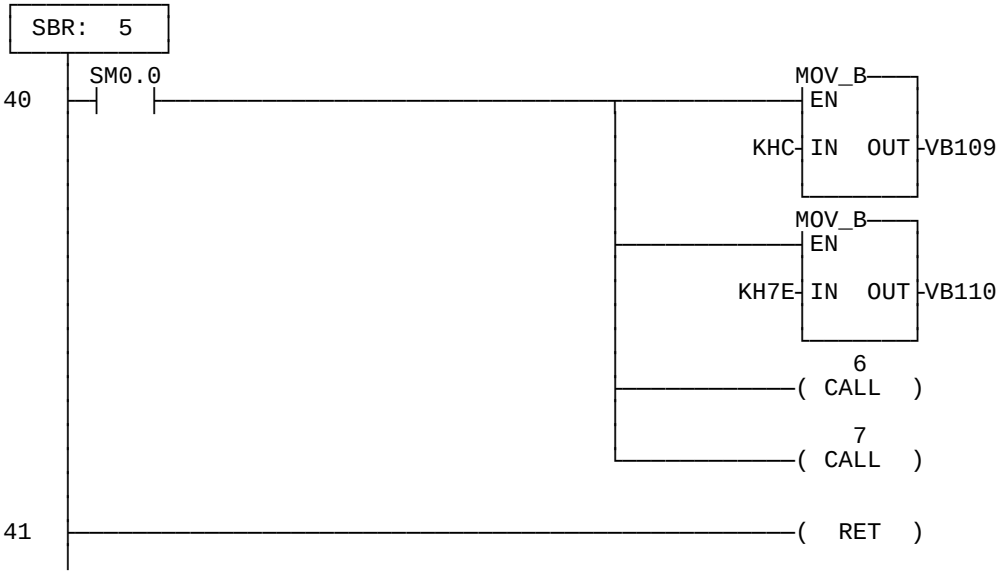

```
LD              M0.1            // If in,
CALL            2               // send msg. w/ increased freq.
RET
```

```
// Subroutine to handle decreasing motor frequency.  Freq. decreased by value
// in VW202.  If I0.5 is ON, it is doubly decremented.  On underflow, freq. set to
// 0.
```



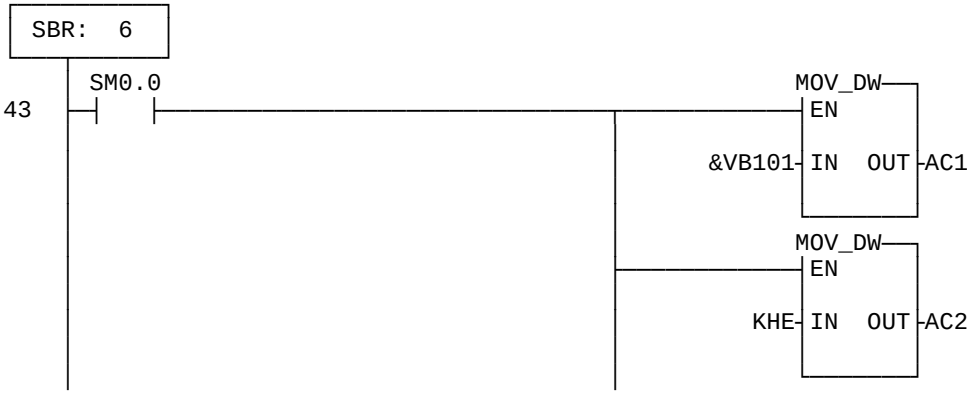

```
SBR         4             //  Decrease freq.
LD          SM0.0         // Always On Bit
-I          VW202,VW200   // by -factor

LD          I0.5          // Double it?
-I          VW202,VW200

LD          SM1.2         // Did we underflow 0?
MOVW        0,VW200       // If so, set to 0.

LD          M0.1          // If in,
CALL        2             // send msg. w/ decreased freq.
RET
```

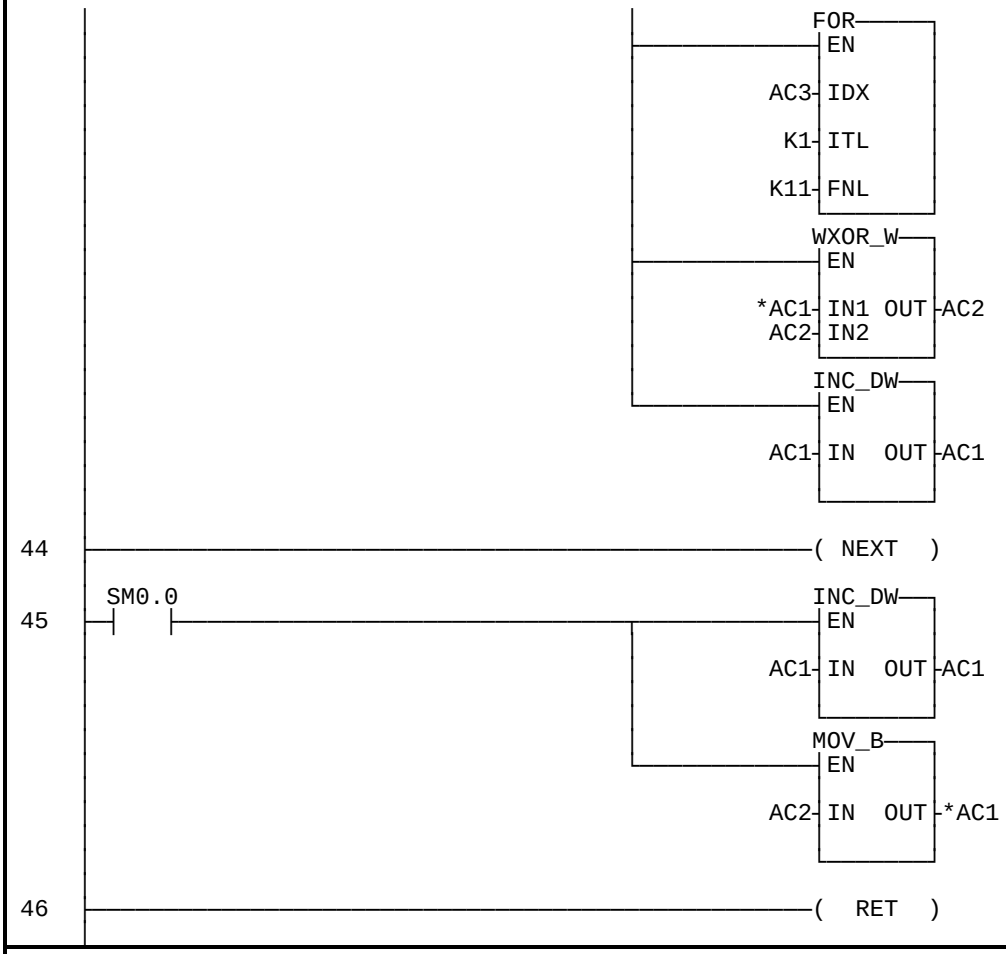
```
// Subroutine to stop motor.
```




```
SBR     5       // Stop motor
LD      SM0.0   // Always On Bit
MOVB    16#0C,VB109   //  Set up STW for
MOVB    16#7E,VB110   //  stop.
CALL    6       // Calculate BCC
CALL    7       //  Initiate/time XMT
RET
```

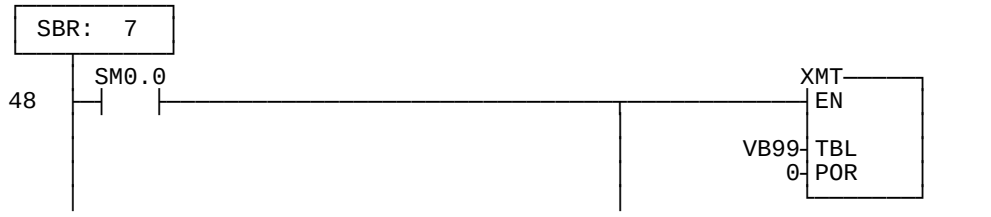
```
// Calculates the XOR block-checksum for the msg. and places it in the buffer.
// HEX Msg. is:  02|0C|ADR|BYTE 1|...|BYTE 10|BCC
// AC1 set to LEN byte(0C) of msg. on entry
// AC1 points to BCC on exit
// BCC has been put into msg. on exit
// AC2 contains BCC on exit
```

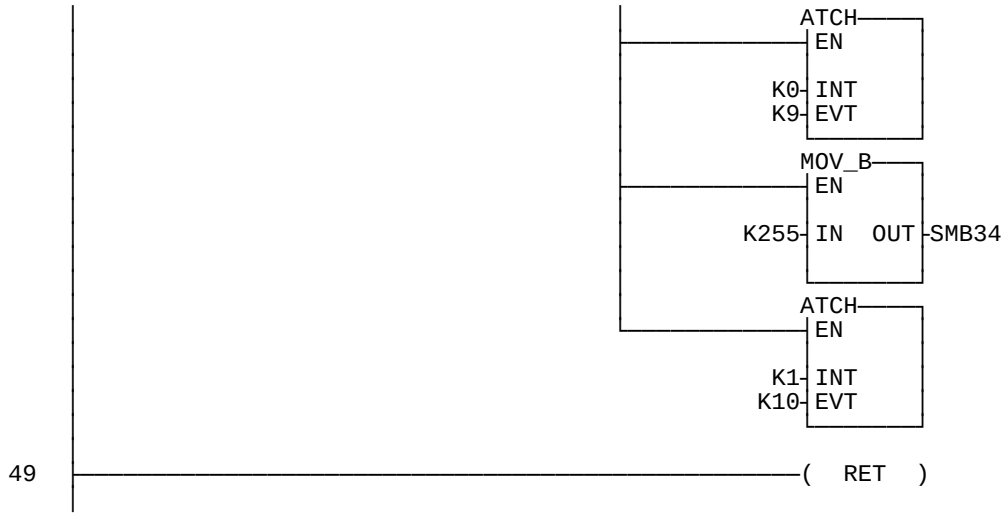





```
SBR         6           // Calculates BCC for USS 5-Word
LD          SM0.0       // Always On Bit
MOVD        &VB101,AC1  //  Set buffer address pointer
MOVD        16#0E,AC2   // 2 XOR 12 - 1st two bytes of msg
FOR         AC3, 1, 11  // Calc. BCC for next 11 chars.
XORW        *AC1,AC2
INCD        AC1         //  Increment address pointer
NEXT
LD          SM0.0       // Always On Bit
INCD        AC1         // Advance pointer to BCC position
MOVB        AC2,*AC1    // Save BCC in msg.
RET
```

// Initiate XMT of msg. & timing of XMT operation

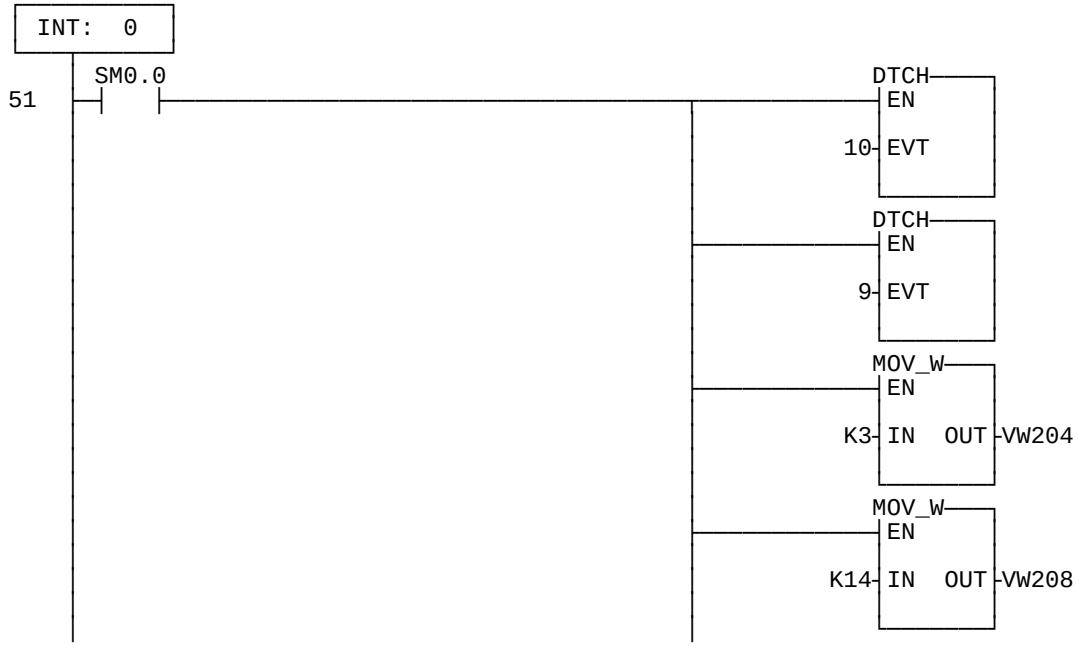


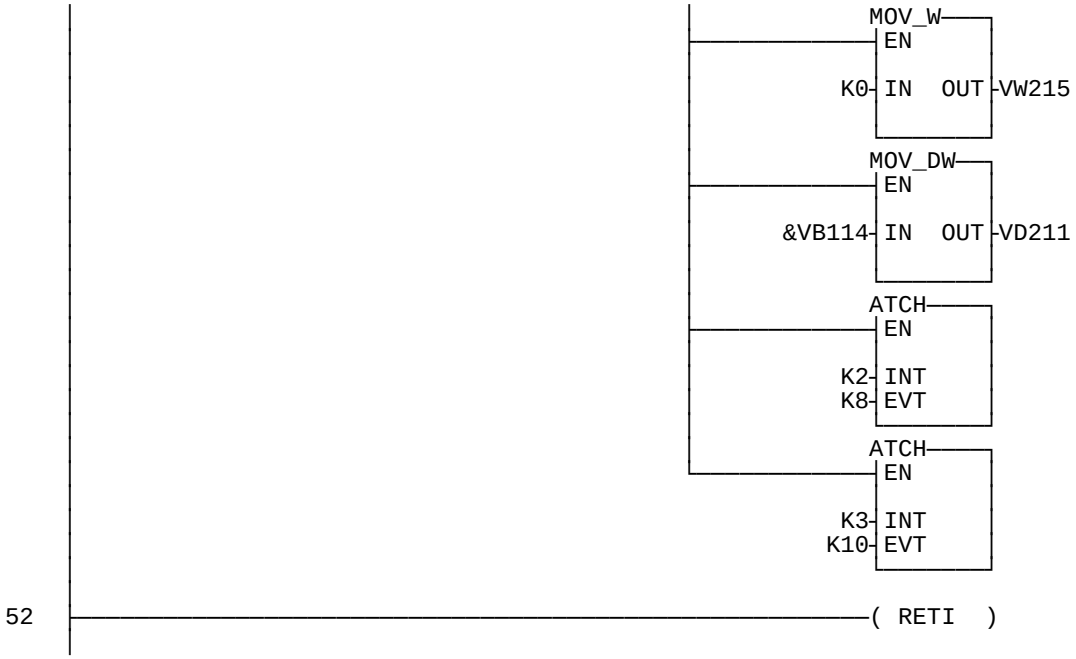



```
SBR     7           // Initiate XMT & capture XMT interrupt
LD      SM0.0       // Always On Bit
XMT     VB99,0
ATCH    0,9         // Catch XMT w/ INT 0
MOVB    255,SMB34   // Set up timing XMT operation
ATCH    1,10        // for 1/4 second.  Should only take about 7ms @ 19.2Kb
RET
```

Interrupt routines

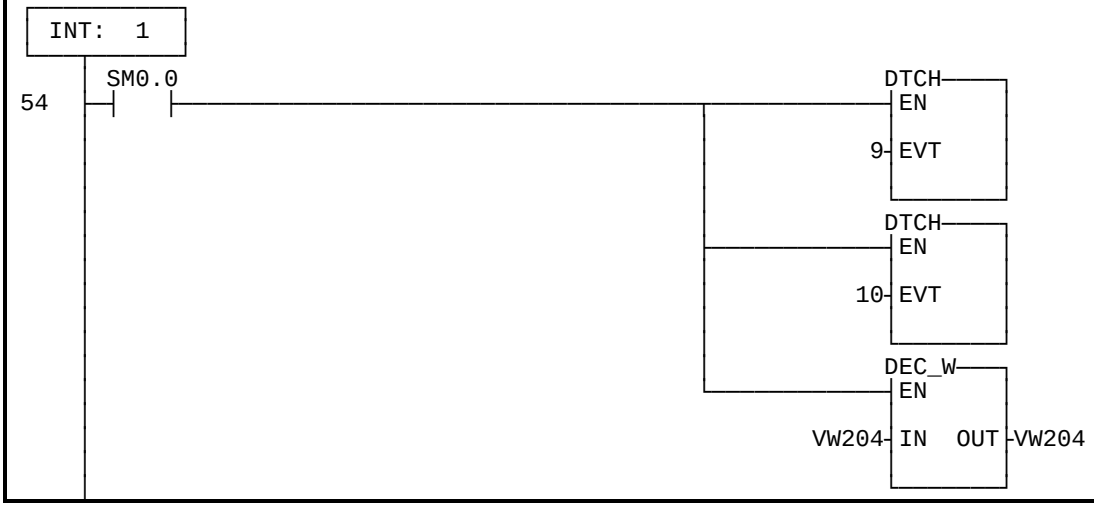
// XMT interrupt handler. Turn OFF XMT timing and set up for receiving slave response.



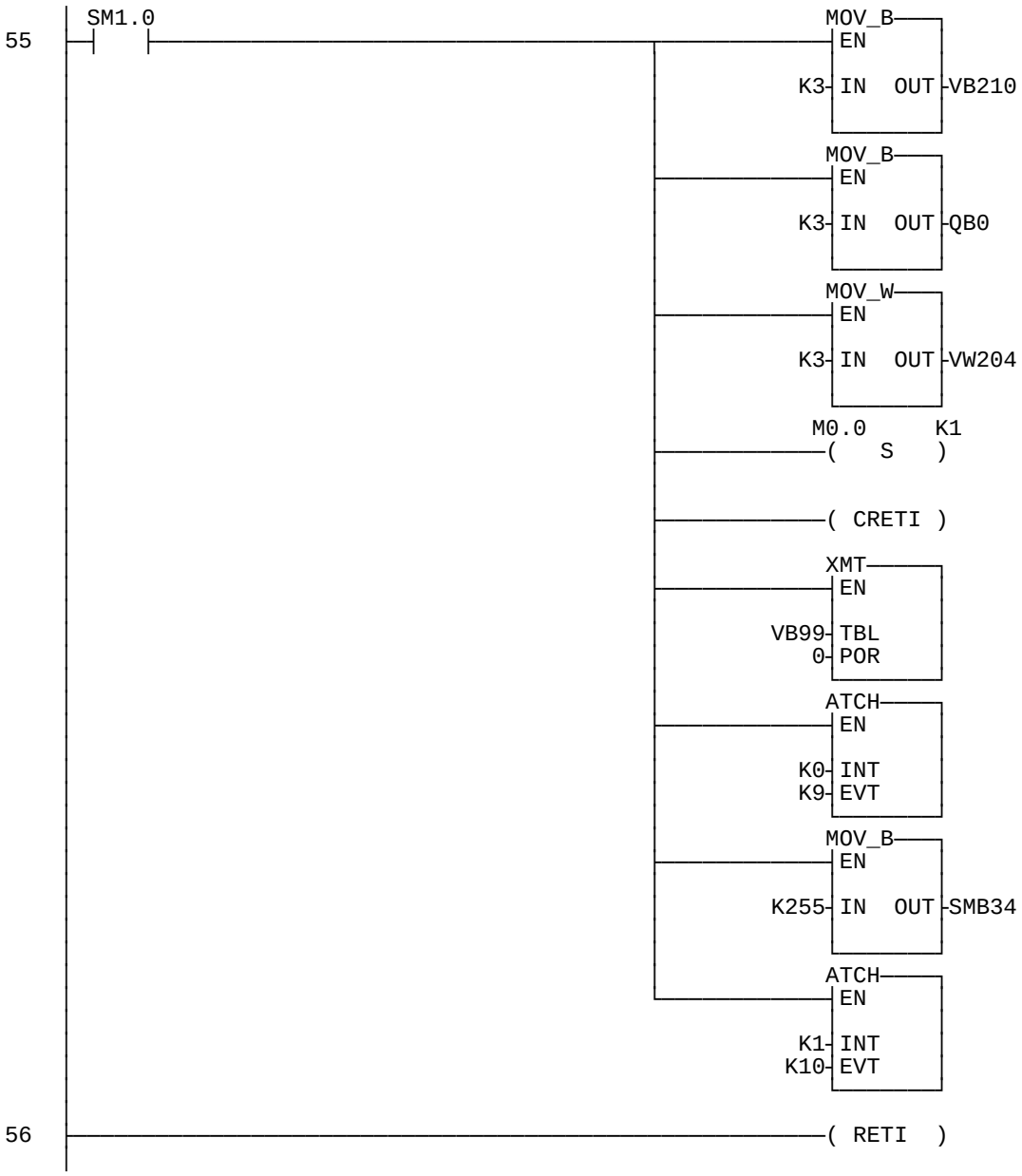



```
INT      0          // XMT interrupt handler
LD       SM0.0      // Always On Bit
DTCH     10         // Quit timing XMT
DTCH     9          // Free XMT event
MOVW     3,VW204    // Refresh XMT retry count
MOVW     14,VW208   // No. chars. to receive in response msg.
MOVW     0,VW215    // Clear BCC accumulator
MOVD     &VB114,VD211 // Set up pointer to receive buffer.
ATCH     2,8        // Catch RCV w/ INT 2
ATCH     3,10       // Start timing receive operation (for 1/4 second)
RETI
```

```
// This handler gains control if XMT times out.  The XMT operation is
// repeated until the XMT retry count is decremented to 0.
```

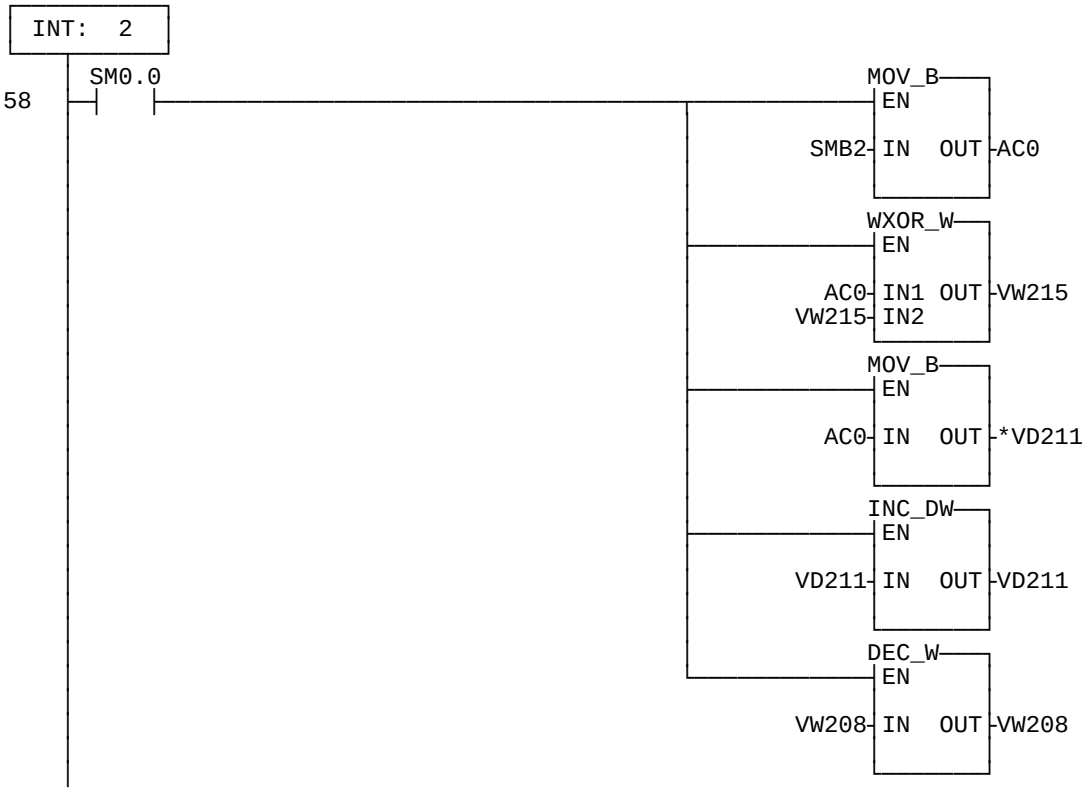


```
INT     1       // Timed interrupt 0 handler - transmit
LD      SM0.0   // Always On Bit
DTCH    9       // Free XMT event
DTCH    10      // Quit timing
DECW    VW204   // Check retry count
```

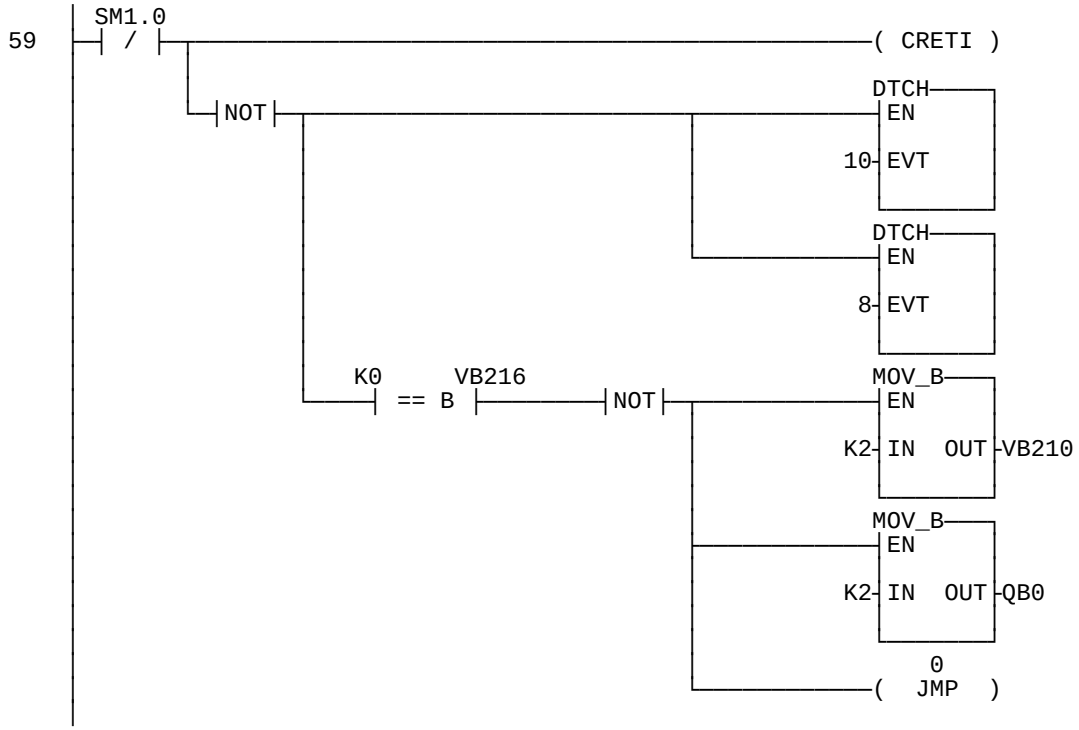


```
LD      SM1.0
MOVB    3,VB210// Indicate transmit timeout
MOVB    3,QB0
MOVW    3,VW204     // Refresh transmit retry count
S       M0.0,1      // Enable RUN, RAMP
CRETI               // Try it again.
XMT     VB99,0
ATCH    0,9         // Catch XMT w/ INT 0
MOVB    255,SMB34   // Set up timing XMT operation
ATCH    1,10        // for 1/4 second.  Should only take about 7ms @ 19.2Kb
RETI
```

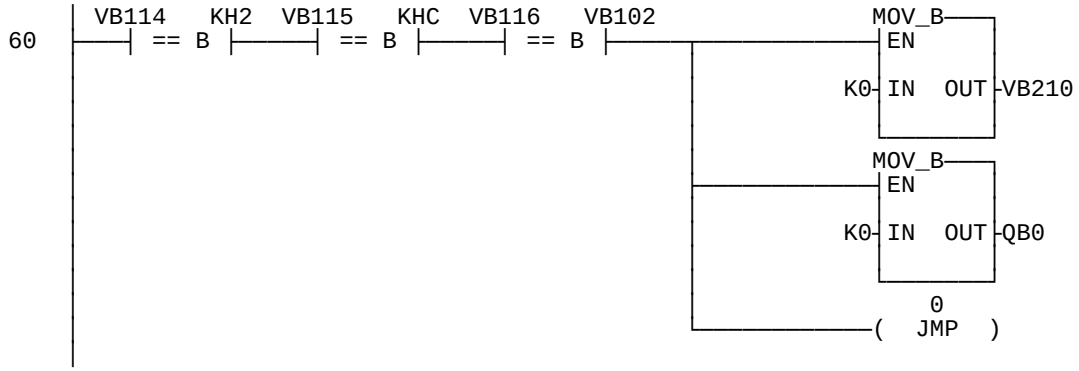
```
// This handler counts characters received & performs error checks.
// If an error was detected, the operation is repeated until the receive retry count
// is decremented to 0.
```



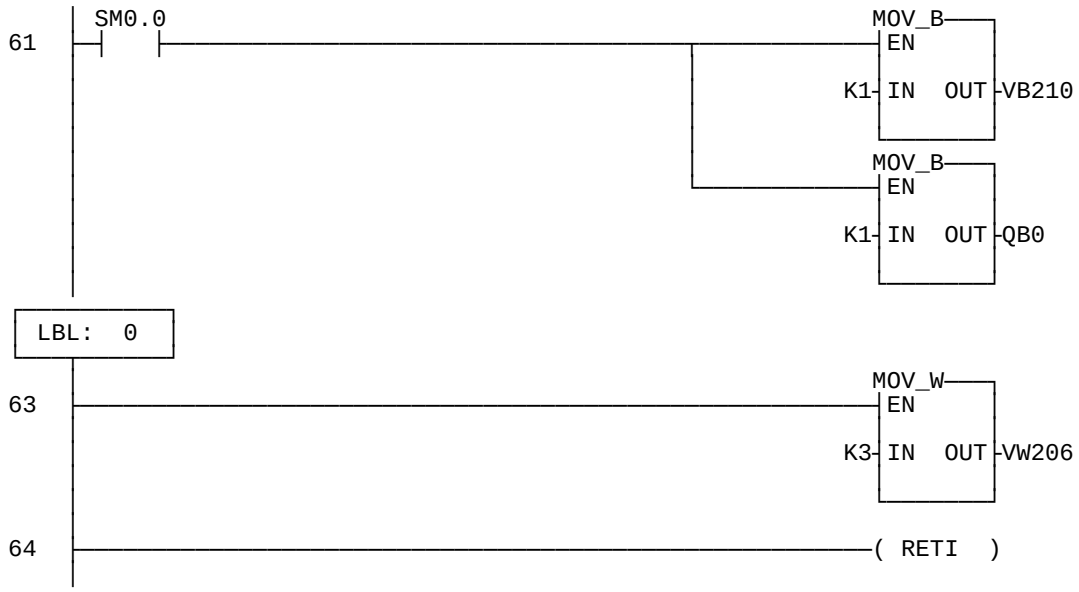

```
INT     2           // Receive char. handler
LD      SM0.0
MOVB    SMB2,AC0    // Get char. received
XORW    AC0,VW215   // Accumulate BCC
MOVB    AC0,*VD211  // Put char. received into buffer
INCD    VD211       // Advance buffer pointer
DECW    VW208       // Decrement no. chars. yet to receive
```




```
LDN     SM1.0     // Check for finished
CRETI
NOT
DTCH    10        // Quit timing receive
DTCH    8         // Turn off receiving
AB=     0,VB216// Check calculated BCC for 0
NOT
MOVB    2,VB210// Bad BCC operation code
MOVB    2,QB0
JMP     0         // BCC OK; check other parts of msg.
```

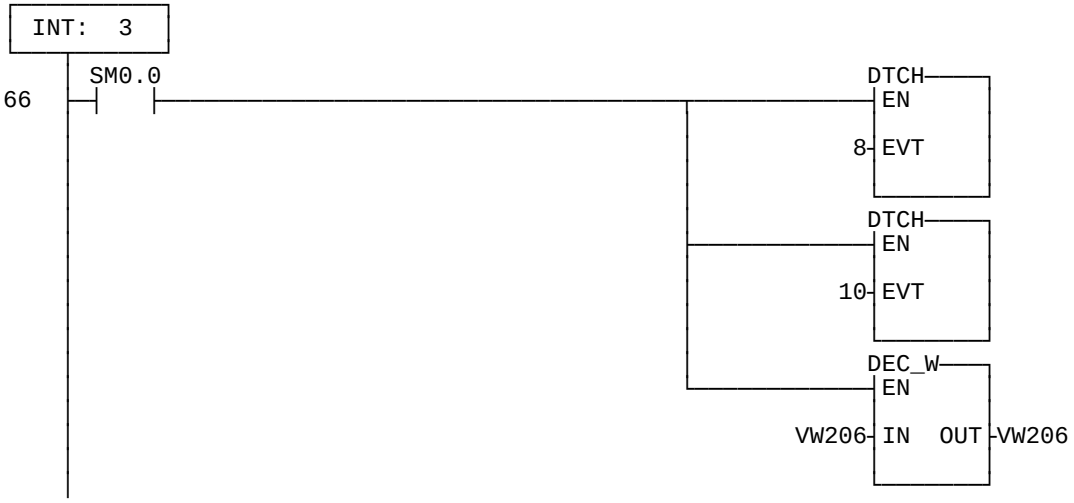



```
LDB=        VB114,16#02    // STX 1st char.?
AB=         VB115,16#0C    // Length = 12?
AB=         VB116,VB102    // Same slave that msg. was sent to?

// Any other specific checks would go here depending upon expected response

MOVB        0,VB210// Operation OK
MOVB        0,QB0
JMP         0
```

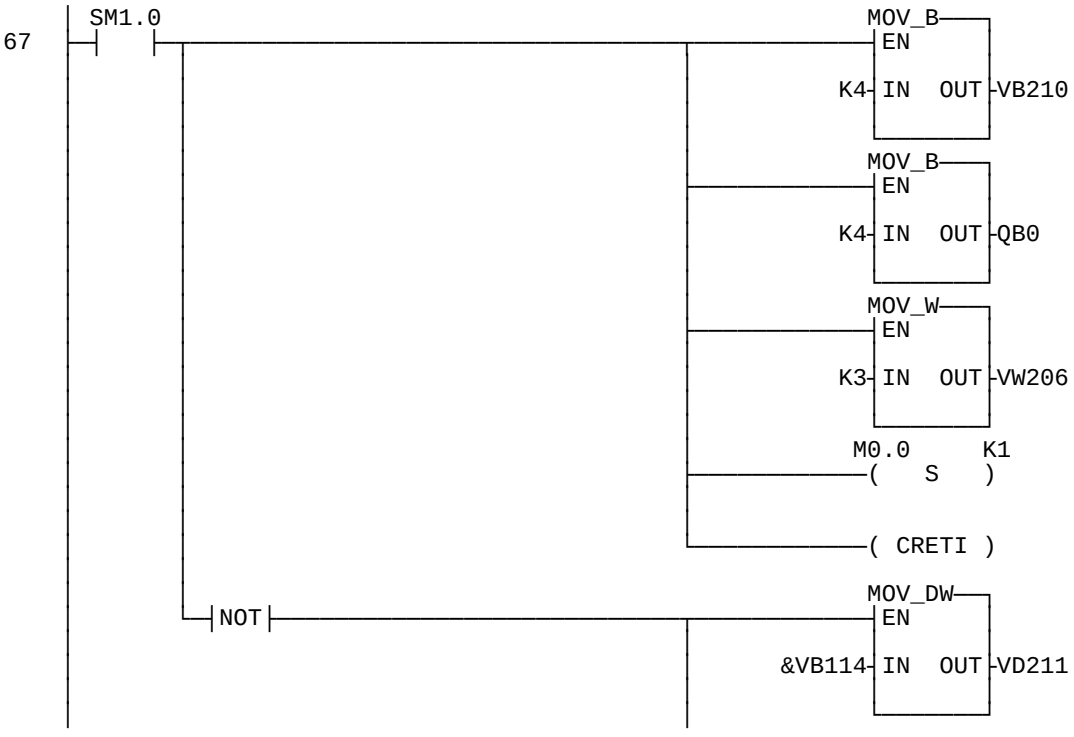


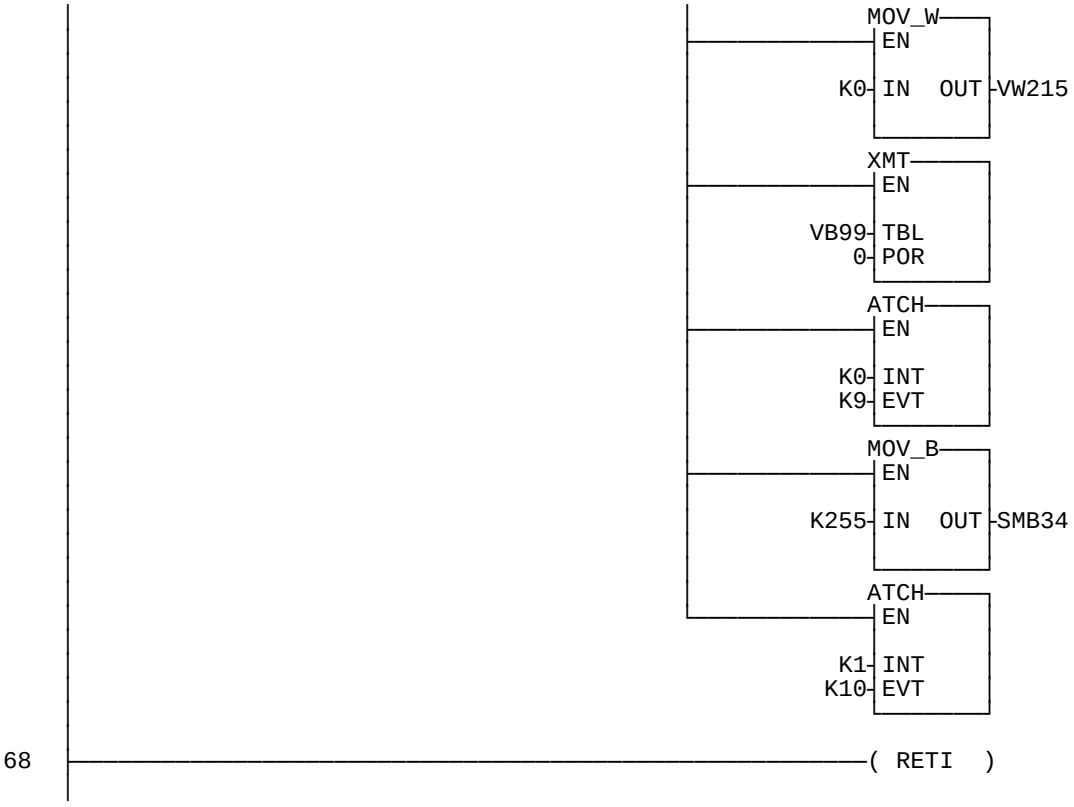

```
LD          SM0.0
MOVB        1,VB210// Something in msg. bad
MOVB        1,QB0
LBL         0
MOVW        3,VW206        // Refresh receive retry count
RETI
```

```
// This handler gains control if the response receive times out.  The msg. is re-sent
// and another receive is attempted.  The operation is repeated on timeout until the
// receive retry count is decremented to 0.
```




```
INT     3       // Timed interrupt 0 handler - receive
LD      SM0.0
DTCH    8       // Free receive event
DTCH    10      // Quit timing receive
DECW    VW206   // Check retry count
```






```
LD      SM1.0
MOVB    4,VB210// Indicate receive timeout
MOVB    4,QB0
MOVW    3,VW206       // Refresh receive retry count
S       M0.0,1        // Enable RUN, RAMP
CRETI
NOT
MOVD    &VB114,VD211 // Set up pointer to receive buffer.
MOVW    0,VW215       // Clear BCC accumulator
XMT     VB99,0        // Re-transmit & try it again.
ATCH    0,9           // Catch XMT w/ INT 0
MOVB    255,SMB34     // Set up timing XMT operation
ATCH    1,10          // for 1/4 second.  Should only take about 7ms @ 19.2Kb
RETI
```

Conversion Notes

To Convert from IEC STL to S7-Micro/DOS STL:

- Add a 'K' before all non-Hex numerical constants (i.e. 4 K4)
- Replace '16#' with 'KH' for all Hex constants (i.e. 16#FF KHFF)
- Commas denote field divisions. Use arrow or TAB keys to toggle between fields.
- To convert an S7-Micro/DOS STL program to LAD form, every network must begin with the word 'NETWORK' and a number. Each network in this Application Tip program is designated by a number on the ladder diagram. Use the INSNW command under the EDIT menu to enter a new network. The MEND, RET, RETI, LBL, SBR, and INT commands each receive their own networks.
- Line-Comments denoted by '//' are not possible with S7-Micro/DOS, but Network-Comments are possible.

General Notes

The SIMATIC S7-200 Application Tips are provided to give users of the S7-200 some indication as to how, from the view of programming technique, certain tasks can be solved with this controller. These instructions do not purport to cover all details or variations in equipment, nor do they provide for every possible contingency. Use of the S7-200 Application Tips is free.

Siemens reserves the right to make changes in specifications shown herein or make improvements at any time without notice or obligation. It does not relieve the user of responsibility to use sound practices in application, installation, operation, and maintenance of the equipment purchased. Should a conflict arise between the general information contained in this publication, the contents of drawings or supplementary material, or both, the latter shall take precedence.

Siemens is not liable, for whatever legal reason, for damages or personal injury resulting from the use of the application tips.