Chapter 5
Event Counting

EVENT COUNTING

Event counting applications find wide use in industrial control, measurement and telecommunications circuits. These applications are characterized by the need to count transitions on some input signal for a specified duration of time.

In a typical Am9513 event counting interface, the input signal is connected to the counter's source input. The counter is gated on for a controlled period of time either through the hardware gate input or by software command. In some operating modes the gate input can be used to save and reset the counter. The counter will usually be programmed to count up and the Load register will usually be set to 0 to facilitate resetting the counter.

Figure 5-1 shows a simple minimum system for event counting. The counter is operated in Mode D, a repetitive counting mode with no hardware gating. The counter's source input is driven by an external signal which is assumed to provide an active-going edge for each event to be measured. The event accumulation process starts when the counter is loaded with zeroes and armed. After the desired measurement interval, the software can disarm the counter to prevent further count accumulation, save and read the accumulated count, or load the counter with 0 to prepare for a new accumulation cycle. The OUT pin of the counter is connected to the RST 7.5 input of the host Am8085A microprocessor in order to generate an interrupt if the maximum count of 65535 binary or 9999 BCD is exceeded. Note that in Mode D, if the maximum count is exceeded, the counter will continue accumulating counts modulo the maximum count. If the counter is operated in Mode A it will stop after exceeding the maximum count until a new ARM command is issued to it.

One limitation of this event counting technique is the need for software timing loops to control the sampling period. Figure 5-2 eliminates this requirement by utilizing level gating control on the event counter and using another counter to control the gate input and hence the sampling period. The event counter is operated in Mode E and the period counter is operated in Mode D or Mode J for either 50/50 or variable duty cycle output waveforms. A 50/50 waveform (Mode D) can be used to alternate between two event counters, one gated with an active-low signal and the other gated with an active-high signal. A variable duty cycle waveform is most useful when only one event counter is used. In either application, a high signal on the period counter's output interrupts the host microprocessor through RST 5.5. The processor's interrupt routine should read the accumulated event count and reset the event counter to 0. An RST 7.5 interrupt will be issued if the event counter reaches the maximum count. If the maximum count is exceeded, the counter will continue to accumulate modulo the maximum count.

The previous example has eliminated the need for software timing loops but still requires the host processor to reset the event counter after each sampling period. This operation can be performed automatically by the hardware gate input if the event counter is operated in Mode O. Figure 5-3 shows the counter interconnections required for this operating mode. The sampling period counter is operated in Mode J and generates an assymmetric output at fixed intervals. Both the low and high times output by the period counter should be longer thant the execution time for a single pass through the software used to perform steps 1 to 4 below. The event counters's gate is driven by this signal. On each active-going gate edge, the event counter's contents are transferred into the Hold register. On the first active-going source edge after the gate edge, the event counter is reloaded from the Load register. The event counter will start counting on the second source edge after application of the gate edge. Because the gate edge controls saving the accumulated count and resetting the event counter, software intervention is not required. The processor can access the results of the most recent event accumulation cycle by setting the Data Pointer and reading the Hold register contents as described in the following steps.

1. Test the period counter's output by reading the Status register.
2. Set the Data Pointer to point to the counter's Hold register. This step loads the Prefetch Latch with the Hold register's contents.
3. Read the Hold register.
4. Test the period counter's output again.

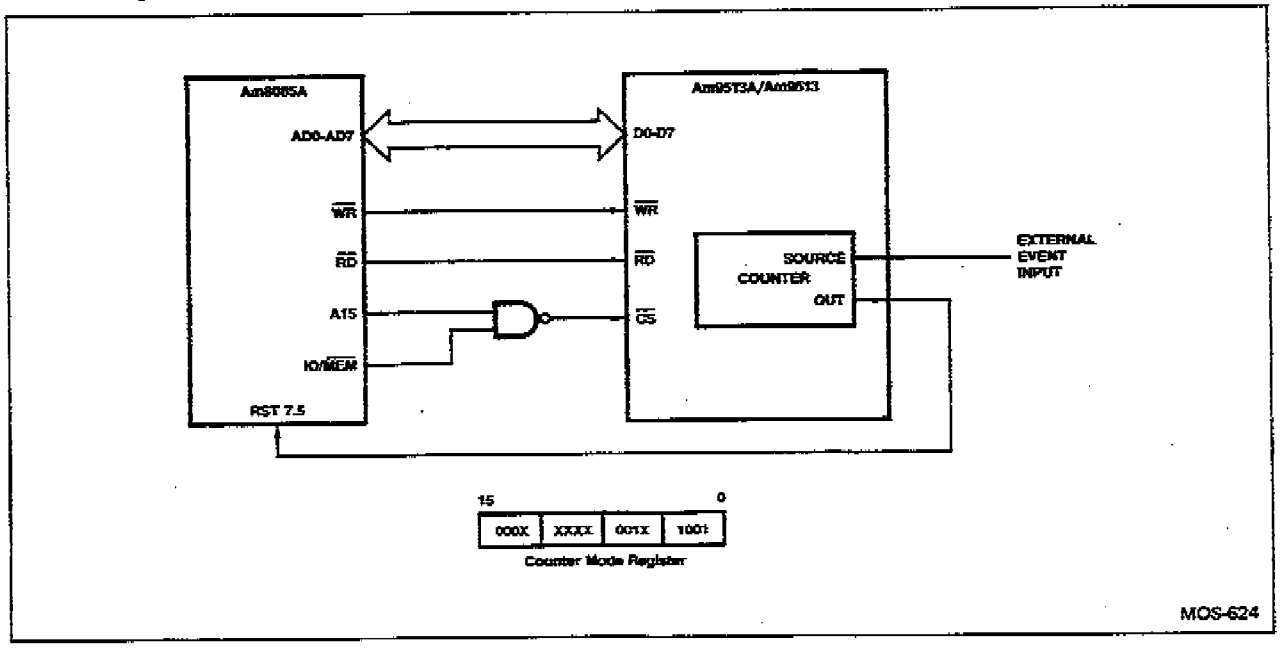

Figure 5-1. Simple One-Counter Event Counting Configuration

If the period counter's output was low in step 1 and high in step 4, an active-going gate edge was applied to the counter during execution of the software. This could potentially cause a faulty read operation for either of two reasons. First, parameters TGVWH and TWHGV in the Am9513 data sheet may have been violated while setting the Data Pointer. See Appendix A for details. Secondly, in an 8-bit bus environment (Master Mode bit MM13 = 0) the gate edge may have split the Hold register values read, with the low byte read reflecting the old Hold register contents and the high byte read reflecting the new updated Hold register contents. Therefore, if the period counter output made a low to high transition between steps 1 and 4, steps 1 through 4 should be repeated. Note that this software must be non-interruptible in order to guarantee that the execution of steps 1 and 4 will be less than half of the period of the period counter's output.

Note that the event counter is reloaded on the first count source edge after application of a gate edge. In most applications, a value of 1 should be reloaded rather than 0, in order to force agreement between the number of count sources applied and the accumulated count. In other words, since the event counter is reloaded by the first source edge, the counter contents after this first pulse should be 1, not 0. Special care must be taken in applications where a possibility exists that no source pulses will be issued to the counter between active-going gate edges. As an illustrative example, consider a counter whose contents are some accumulated value K. When a gate edge is applied to the counter, this value K is transferred to the Hold register. The counter contents at this point are unchanged (i.e., are equal to K). In a normal count cycle, on the next source edge, the counter would be reloaded from the Load register and would start counting events. Consider the situation where no source edges occur before a second active-going gate edge is applied. The counter contents are still K since no reload has occurred, and the Hold register will once again be set to K. A value of K has been saved, even though no source pulses were applied during the sampling period. It can be seen that proper operation of this gate-initiated save/reload operation requires a minimum of 1 active-going source edge between active-going gate edges. Some applications will be able to guarantee this requirement of at least one source edge per sampling cycle by virtue of the signal being measured. Other applications can avoid reading false values by issuing a LOAD command to the counter and then reading the Hold register after at least two sampling periods have elapsed. If the Hold register contents equal the Load register contents, no source pulses are being received by the counter.

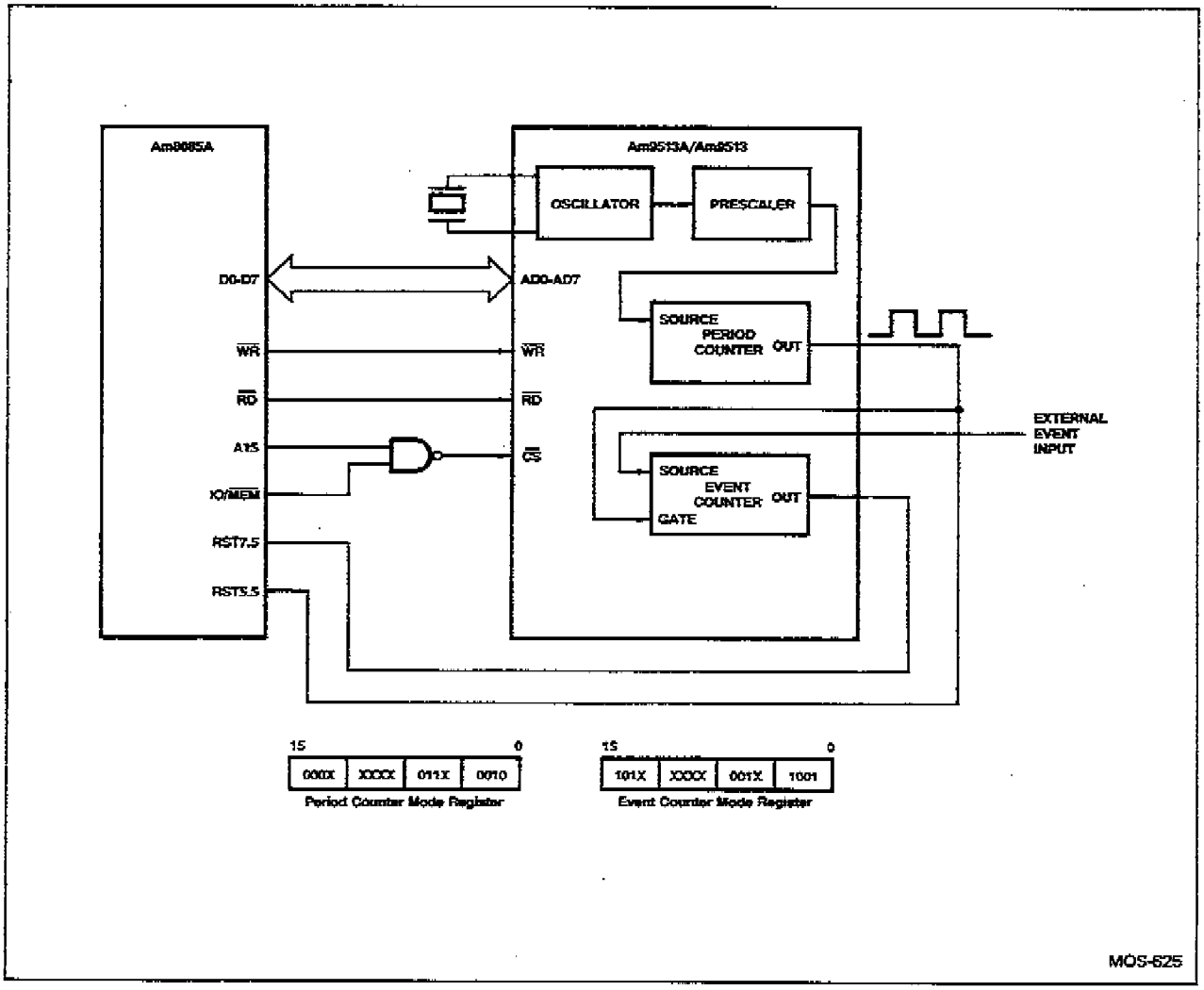

Figure 5-2. Event Counting with Hardware Gating

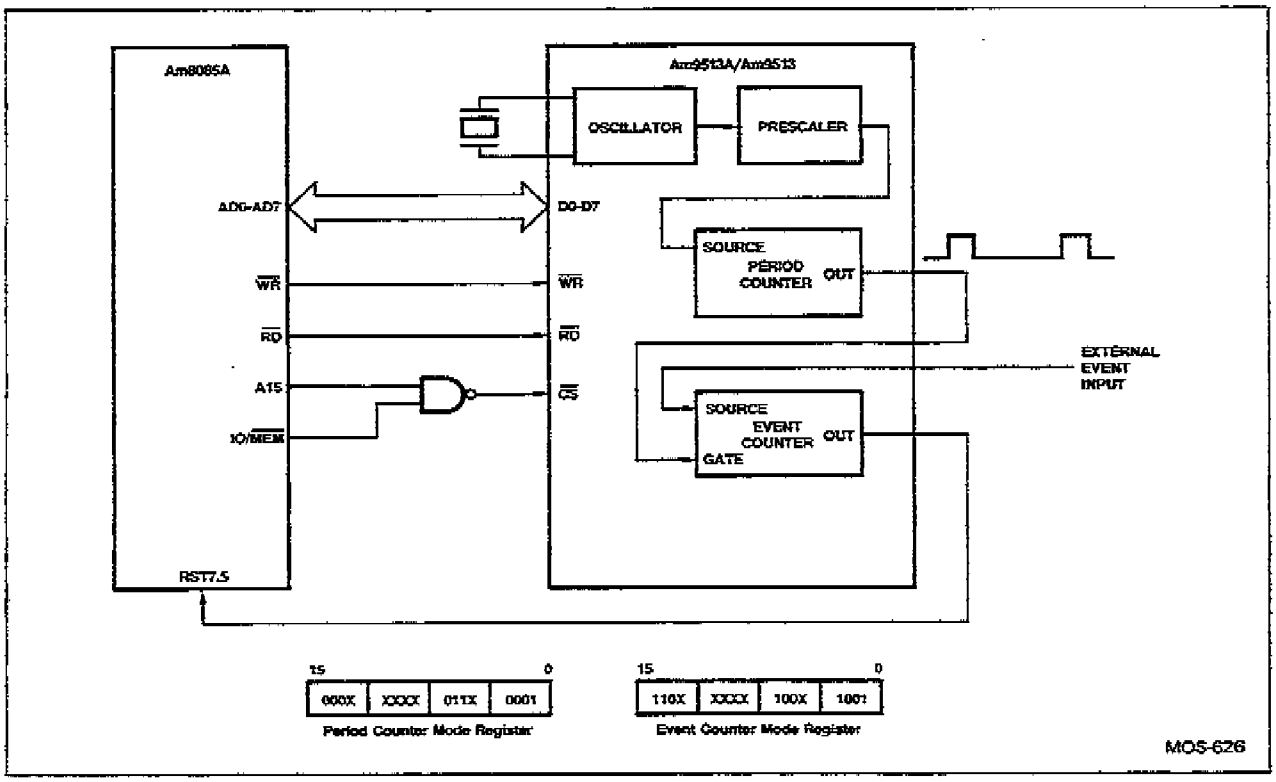

Figure 5-3. Event Counting with Hardware Read and Reset

INDUSTRIAL CONTROLLER – A Z8000 EXAMPLE

A certain manufacturer of widgets has provided the following part specification for the large-volume supply of a microprocessor-based controller system:

"The handling system is constructed around a chain conveyor. Unsorted widgets are loaded onto one end of the conveyor at non-equal intervals, transported along and unloaded at the far end for shipping. Widgets without an identifying mark must be knocked off the conveyor by an existing hammer mechanism, previously manually activated. A small area is available to inspect the widget for the identifying mark a short distance from the reject hammer. The identifying mark fluoresces under the action of ultra-violet illumination of a specific wavelength, otherwise the local area will remain in darkness. The identifying mark, if present, is guaranteed to appear within a defined area."

"The chain conveyor speed is variable within a small known range. Widgets are presented such that a widget does not approach the inspection area until after the preceding widget has left the rejection point. To achieve compatibility with current manual inspection the action of the hammer must be described in terms of chain links. The hammer lies A links beyond the inspection point and must be activated for B links duration to achieve correct rejection. The hammer should not be active outside this period. Sensing of chain conveyor speed must be achieved by non-contact means due to safety regulations."

"Readouts must be provided of current total reject number and the number of rejects per hour. The readings will be reset every 8 hours or so. Widget throughput is around 600 per hour."

Minimizing the component count prompted the system designers to use a single Am9513 device to provide all the counting and timing functions required. The implementation is illustrated in Figure 5-4. The following points discuss the various areas in more detail.

1. Chain Speed Sensor. Unfortunately, due to the chain construction, all available non-contact sensors produce a noisy output in addition to an identifiable edge at the start of each link. A mask pulse is thus required of known maximum duration since the chain speed is known to lie over a small range. Calculations show that synchronization will always be achieved within 20 links, an acceptable power-up time.

   This function is realized by operating counter 3 in mode F. An active edge on the link sensor enables counter 3 to count down from the value stored in the Load register, further gate inputs (link sensor outputs) being ignored until the end of count, or TC. At TC the counter waits for a new gate input to repeat the cycle, in normal operation the commencement of the next link. The clock input to counter 3 is defined to be a suitable internal source, depending on the chain conveyor speed. The TC output of counter 3 cycles at once per link, generating the "link clock" for the system.
2. Widget Mark Sensor. Due to the large area within which the fluorescent mark may appear, the optical sensor produces a fairly slow pulse in the presence of noise and 60Hz and 120Hz interference. For this reason, the sensor output is routed via a high-gain squaring amplifier to an integrator which is reset at 17msec intervals. The integrator cancels most of the interference while providing a clear output of a sensed mark. This output is gated as the integrator reset is applied, generating a single reject pulse if the mark is identified.

   The reset clock is provided by the FOUT output, driven from an appropriate internal source.

3. Hammer Pulse. To achieve correct operation of the hammer, a delayed one-shot is required, triggered from the gated reject pulse. Counter 4 is set to operate in mode L and the reject pulse applied to the gate input, thus starting the count. The counter 4 clock input is programmed to TCn-1, the "link clock" output produced by counter 3. The Load register contains the value A, and the Hold register contains the value B. Counter 4 TC then provides the hammer output of duration B links after a delay of A links.
4. Time-of-Day Clock. A suitable clock is provided by counter 1 and counter 2 (see time-of-day clocking description).
5. Reject Accumulator. The spare counter 5 is used as a reject accumulator, finally freeing all CPU intervention with the Am9513 operation save for periodic readings to update the displays, etc. Counter 5 is operated in mode A as a simple counter. The source input is programmed to be TCn-1, which is the hammer output of counter 4 providing one count per reject. If counter overflow occurs then the returned count will be the load register value, nominally zero.

Notice that the entire implementation uses only four external connections to the Am9513. The listing in Figure 5-5 provides a Z8000 Assembly implementation example.

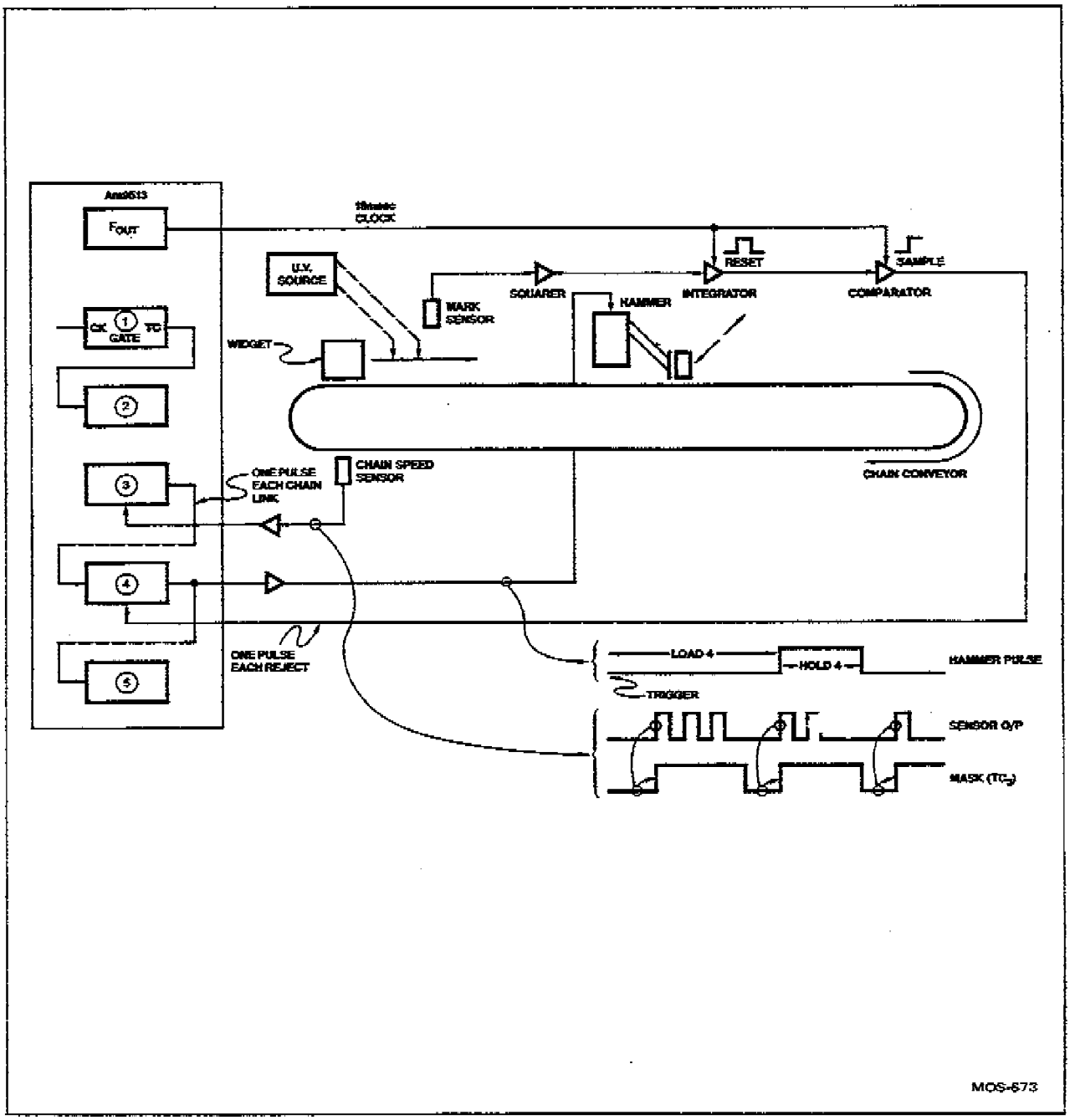

Figure 5-4. Am9513 Industrial Controller

```
MODULE "WIDGET";

%       Coded for AmZ8000

%       This file provides a short industrial controller example using the
% Am9513 System Timing Controller as a multi-purpose timing and counting
% element.

%       Warning: Due to the nature of this file it has not been fully tested.

%       This file is part of the Am9513 Software Applications Manual.
% is not copyright and you can store it on what you like.


CONST
A_DATA      =    0FFD8H          , % Am9513 data port
A_CTRL      =    0FFDAH          , % Am9513 control port
CHAIN_MASK  =    5000            , % Mask pulse for chain sensor noise
A_LINKS     =    42              , % Activate hammer a_links after inspection
B_LINKS     =    7               ; %  ... for b_links duration


INCLUDE 'B:EQUSZ.ZSC'            ; % Listed in Appendix 7
INCLUDE 'B:Z8000MAC.ZSC'         ; % Listed in Appendix 5

ORIGIN  9000H ;


WIDGET:
        % Reset and initialise the Am9513.

        RESET                    ; % Reset, Load all, set 16 bit bus, set data ptr

        MASTER TOD_100HZ,DISABLE,DISABLE,F5,10,ON,BUS_16,ON,BCD ;

        CALR SETTIME             ; % Get the time of day clock set up

        MODE_REG 3,ACT_HI_TC,DOWN,BCD,MODE_DEF,F4,RISE,EE_GATE_N ;
        LOAD_REG 3,CHAIN_MASK    ; % Chain speed sensor

        MODE_REG 4,TC_TOGGLE,DOWN,BCD,MODE_JKL,TC_NM1,RISE,HE_GATE_N ;
        LOAD_REG 4,A_LINKS       ;
        HOLD_REG 4,B_LINKS       ; % Reject hammer output

        MODE_REG 5,OFF_OC_TC,UP,BINARY,MODE_ABC,TC_NM1,RISE,NO_GATE ;
        LOAD_REG 5,0             ; % Widget reject counter

        CLEAR    3               ; % TC outputs must start low for sync
        CLEAR    4               ;
        ARM      1,2,3,4,5       ; % Time and action starts now
        RET                      ;
```

Figure 5-5. AmZ8000 Assembly Listing for Controller

```
SETTIME:
        % Set up channels 1 and 2 to time of day mode

        MODE_REG 1,OFF_OC_TC,UP,BCD,MODE_DEF,F5,RISE,NO_GATE;

        MODE_REG 2,ACT_HI_TC,UP,BCD,MODE_DEF,TC_NM1,RISE,NO_GATE;

        LOAD_REG 1,Ø            ; % Ø here is a constant
        LOAD_REG 2,Ø            ;

        LOAD     1,2            ; % Enable TOD counting

        LOAD_REG 1,TIME(2)      ; % Current time value - notice no I flag reqd.
        LOAD_REG 2,TIME         ;

        LOAD     1,2            ; % Set the current time

        LOAD_REG 1,Ø            ;
        LOAD_REG 2,Ø            ; % Recall repeat reload values needed

        RET                     ;


READ_IT:
        % Routine to get current system status.
        %
        % Exit: R1    holds time XX:XX (hrs:min) in 4 x BCD format
        %       R2    holds current total reject #

        SAVE    1,2,5           ; % Fix the data
        POINT   1,HOLD          ;
        IN      RØ,A_DATA       ; % Read secs, tenths
        TEST    RØ              ;
        JR      NZ,READ_IT1     ; % Read was ok

        SAVE    2               ; % We might have missed a carry bit
READ_IT1:
        POINT   2,HOLD          ;
        IN      R1,A_DATA       ; % Read hrs, mins
        POINT   5,HOLD          ;
        IN      R2,A_DATA       ; % Read current reject total
        RET                     ;

                        % Required start time hrs, min, secs, tenths

TIME:   BYTE:   09H,15H,Ø,Ø     ;  % About the time we get to work (9:15am)

END.


A>
```

Figure 5-5. AmZ8000 Assembly Listing for Controller (Cont.)