AN1226

Use of the 68HC705C8A in Place of a 68HC705C8

By Russ Walin
CSIC Product Engineering
Austin, Texas

Introduction

This application note is intended to document the differences between the 68HC705C8A and the 68HC705C8. It will also describe uses for the "A" features (new features per customer requests), which include the port B keypad interrupt/pull-ups, 68HC05C4A-type COP, and the high current drive on port C. The pull-ups and C4A-type COP can be enabled with two additional mask option registers (MOR).

Background

The 68HC705C8A is an enhanced version of the 705C8. It is designed to be a drop-in replacement for the 705C8. There are some inherent differences that the user should be aware of such as the port C7 current drive characteristics, the MOR programming requirements, and the geometries used in manufacturing. A bug with the SPI on the 705C8 was fixed and the boot ROM code was changed.

Using the 705C8A in Place of a 705C8

When using the 705C8A in place of a 705C8, note the following points.

1. The most significant difference that exists when using the 705C8A as a replacement for the 705C8 is the output current drive capability on the port pin PC7 (port C bit 7) on the 705C8A. The drive current was increased to provide LED drive capability on PC7. The output drive characteristics of PC7 for both the 705C8 and the 705C8A are shown in **Figure 1**. There is no way to reduce the PC7 drive current of the 705C8A to emulate the 705C8.

Characteristic	MC68HC705C8	MC68HC705C8A
V_{DD} = 5.0 V ± 10% PC7 current drive (I_{OH}) @ V_{OH} = V_{DD} –0.8 V PC7 current sink (I_{OL}) @ V_{OL} = 0.4 V	 0.8 mA 1.6 mA	 5.0 mA 20.0 mA
V_{DD} = 3.3 V ± 10% PC7 current drive (I_{OH}) @ V_{OH} = V_{DD} –0.3 V PC7 current sink (I_{OL}) @ V_{OL} = 0.3 V	 0.2 mA 0.4 mA	 1.5 mA 6.0 mA

2. Two additional MOR registers have been added on the 705C8A. To emulate the 705C8, the MOR1 and MOR2 must not be programmed (i.e., the port B interrupts/pull-ups and C4A COP will not be enabled). The erased state of the 705C8's EPROM is "0", so the default is "A" features disabled. That is, $00 must be programmed into locations $1FF0–1FF1. On the 705C8 locations, $1FF0–1FF1 are EPROM bytes that are reserved for test. Depending on the programmer used, these bytes on the 705C8 may or may not be programmed. For example, the Bootloader board, described in *MC68HC705C8 Technical Data* (MC68HC705C8/D), programs the EPROM bytes at $1FF0–1FF1, so the master EPROM (2764) must contain $00 at locations $1FF0–1FF1.

3. Programming characteristics should be similar to the 705C8. The programming voltage V_{PP} for the 705C8A should be V_{PP} = 14.5–15.0 V, as with the 705C8. The programming times also remain unchanged at 2 ms/byte.

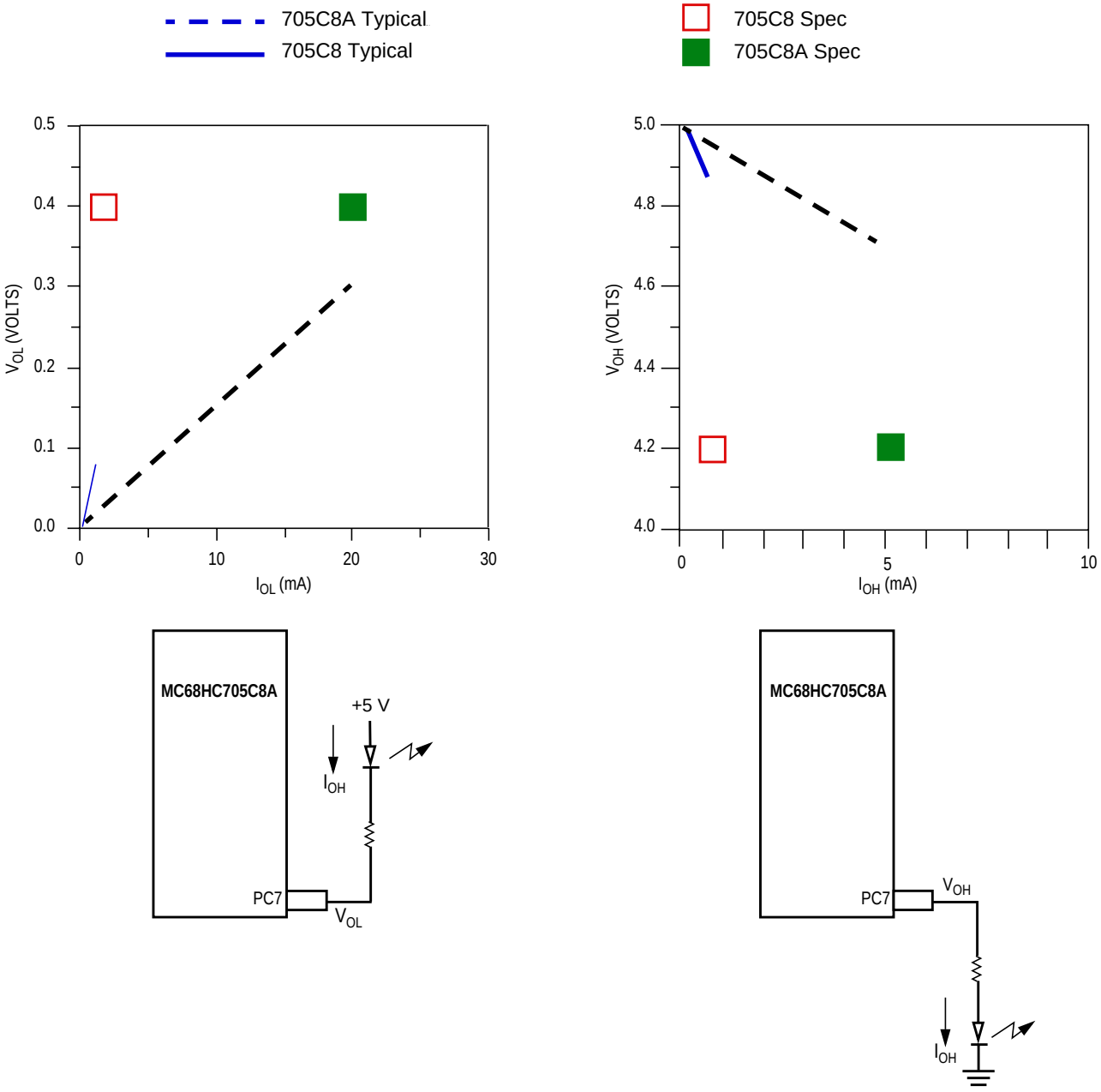

Figure 1. PC7 V_{OL}/I_{OL} and V_{OH}/I_{OH} Comparison of the 705C8 and the 705C8A V_{DD} = 5.0 Volts

4. The 705C8A is made with 1.2 micron CMOS technology whereas the 705C8 is made with 1.75 micron technology. The operating I_{DD}, wait I_{DD} and stop I_{DD} of the 705C8A are all similar to 705C8, and the maximum I_{DD} specifications remain unchanged.
5. A bug with the 705C8 SPI has been corrected on the 705C8A. When the 705C8A SPI is in slave mode with CPHA = 1 and CPOL = 0, the SPIF bit occasionally will not become set, wrongly indicating an incomplete transmission. This problem was corrected on the 705C8A. Because of this, the SPI slave mode enable lag time is larger than that of the 705C8. The SPI slave mode enable lag time is RATE * 1.5 (RATE = 1/frequency).
6. The code in the Boot ROM location $1F00–$1FEF has been changed. The 705C8A contains Boot 7C8A Rev. 3.0 code.
7. The function of several bits in the PROG registers ($1C) have changed. These bits were not implemented on the 705C8. On the 705C8A, bits 1 and 3–7 are implemented as test bits. The bits should always be written as "0".

   This change will affect programmer manufacturers and users who use their own programming algorithm.

	Bit 7	6	5	4	3	2	1	Bit 0
$1C	U	U	U	U	U	LAT	U	PROG

U = Unused

Figure 2. 705C8 PROG Register

	Bit 7	6	5	4	3	2	1	Bit 0
$1C	T	T	T	T	T	LAT	T	PROG

T = Reserved for test. Always write"0".

Figure 3. 705C8A PROG Register

Using the Additional "A" Features of the 705C8A

The 705C8A has several features added due to customer requests, which are referred to as the "A" features. These features include the C4A-type COP, the port B interrupts/pull-ups, and the LED drive capability of PC7.

The C4A-type COP is similar to the 705C8 with the exception of the timeout period, which is fixed at $(1/f_{OSC}) * 2^{18}$ (see Figure 2). The C4A-type COP is implemented with an 18-bit ripple counter. It has a timeout period of 64 milliseconds at a bus rate of 2 MHz. This COP is intended for use with the emulation of a 68HC05C4A. The C4A-type COP is enabled by programming the EPROM bit 0 (NCOPE) at address $1FF1 to a "1". If COP times out, a system reset will occur. The COP is cleared by writing a "0" to the PBPU0/COPC bit (bit 0) at location $1FF0. Reading location $1FF0 will return the contents of MOR1. Location $1FF0 is also the address used to enable the pull-up resistors; however, writing a "0" to reset the COP will not have any effect on the state of the pull-ups.

The 705C8-type COP is implemented as part of the 16-bit timer. To enable the 705C8-type COP, the C4A COP should be disabled (EPROM bit 0 at location $1FF1 should not be programmed). The 705C8-type COP is enabled by setting the COPE bit (bit 2 at location $001E) to a "1". The 705C8-type COP uses 11 bits of the 16-bit timer, which includes a /4 fixed prescalar. This yields a 2^{15} divide by (see **Figure 4**). There are three other options for the timeout period that are determined by the state of CM0/CM1 (bit 1 and bit 0 of the COP control register $1E). The timeout period for various timeout periods is shown in **Table 1**.

The COP is reset by writing a $55 to the COP reset register at $1D, and then writing an $AA to the COP reset register. For more information on the 705C8-type COP, see **3.1.3 Computer Operating Properly (COP) Watchdog Timer Reset** in *MC68HC705C8 Technical Data* (MC68HC705C8/D).

The 705C8A also contains interrupts and pull-ups on port B intended for implementing a keypad. The pull-up simplifies hardware needed for a

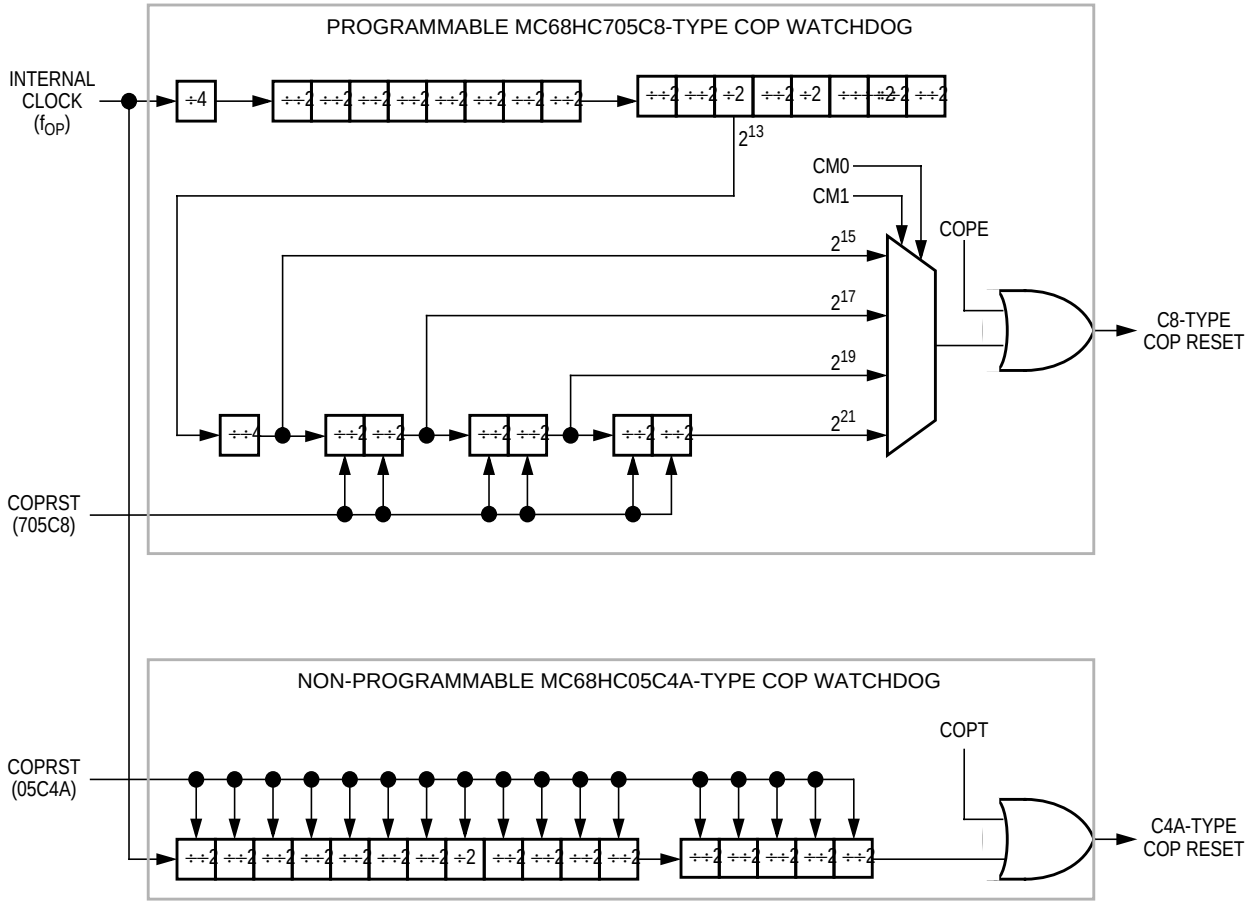

Figure 4. COP Block Diagram of the MC68HC705C8A COP Watchdogs

Table 1. Comparisons of 705C8 versus C4A COP Timeout Periods

COP Type	CM0	CM1	$f_{OSC}/2^{15}$ Divided by	f_{OSC} = 4.0 MHz f_{OP} = 2.0 MHz	f_{OSC} = 3.5795 MHz f_{OP} = 1.7897 MHz	f_{OSC} = 2.0 MHz f_{OP} = 1.0 MHz	f_{OSC} = 1.0 MHz f_{OP} = 0.5 MHz
705C8	0	0	1	16.38 ms	18.31 ms	32.77 ms	65.54 ms
705C8	0	1	4	65.54 ms	73.24 ms	131.07 ms	262.14 ms
705C8	1	0	16	262.14 ms	292.95 ms	524.29 ms	1.048 s
705C8	1	1	64	1.048 s	1.172 ms	2.097 s	4.194 s
C4A	NA	NA	NA	65.54 ms	73.24 ms	131.07 ms	262.14 ms

keypad by eliminating the need for pull-ups externally. The interrupt capability simplifies the software by eliminating the need to poll port lines or wire-ORing the lines to the IRQ pin to detect that a key has been pressed. A diagram of a simple keypad is shown in **Figure 5**. The pull-ups are enabled by programming the corresponding bit in the MOR1 at location $1FF0 when programming the user code in the main EPROM. Once the pull-ups have been enabled, it is not possible to disable them without UV erasing the EPROM.

The following is an example of how to use the port B interrupt and pull-ups to implement a keypad. The MOR1 byte must have a $0F programmed into it to enable the interrupts and the pull-up. The example first sets ports A and C as output and writes a $00 to port A and a $55 to port C. Port C is used to show which row and column to indicate the key that was pressed. Port A is incremented every time a key is pressed and will register if bounce caused any extra interrupts. The set-up also configures the interrupt for edge- only, drives PB7–PB4 low and configures them as outputs. Once the 705C8A has been properly configured, the interrupt mask bit is cleared and the stop instruction is executed to save power while the 705C8A idle.

Once a key is pressed, the part exits stop mode and branches to the interrupt service routine as indicated with the IRQ vector. This routine waits 30 ms to prevent bounce and decodes which row and column the closed key is in.

Care must be taken to allow enough time once a column has been deselected before reading the state of the rows. The pull-ups are fabricated with a weak P-channel device. The IV characteristics are shown in **Figure 6**. Since they are such small devices, there is a long rise time for a port B input to be pulled high after the column is no longer driven low. Thus the delay is added just prior to reading the state of the rows.

Once the column and row have been decoded, the routine enters a loop to hold until the key is released. Again, a 30 ms pause is executed to prevent bounce. The code then writes the row and column out to port C and increments port A to indicate another key stroke. The service routine ends with an RTI and the code branches back to the main routine where it reconfigures port B to wait for the next key stroke.

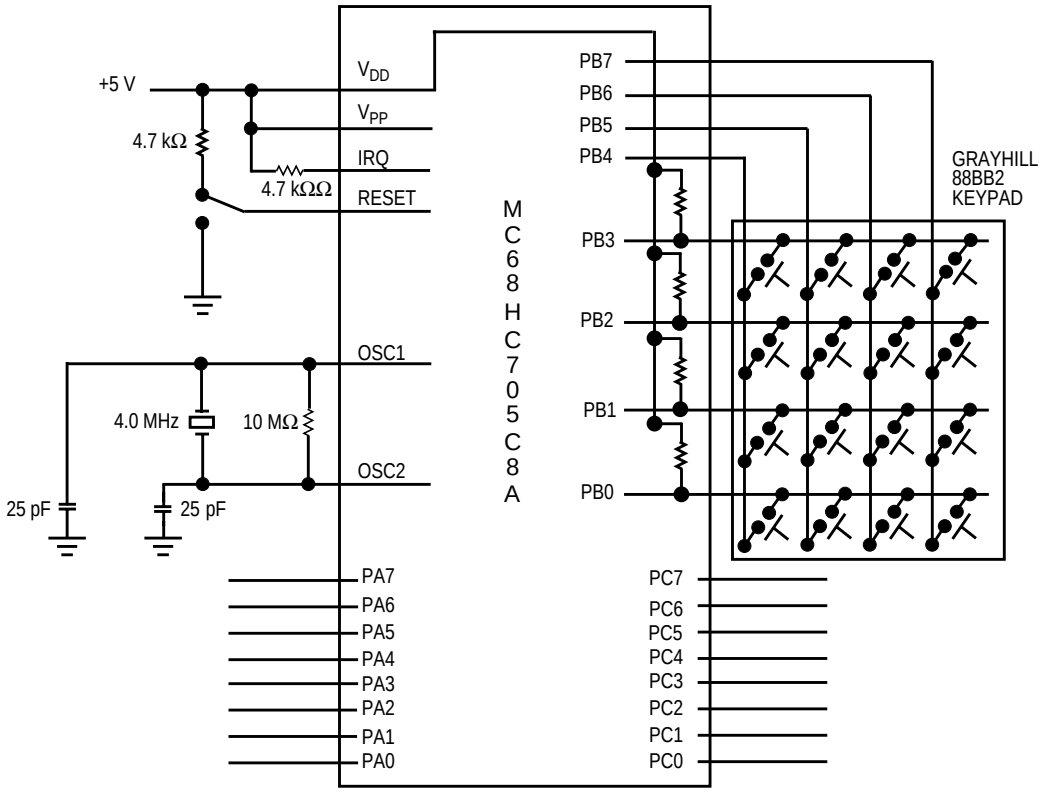

Figure 5. Example of a 4 x 4 Keypad

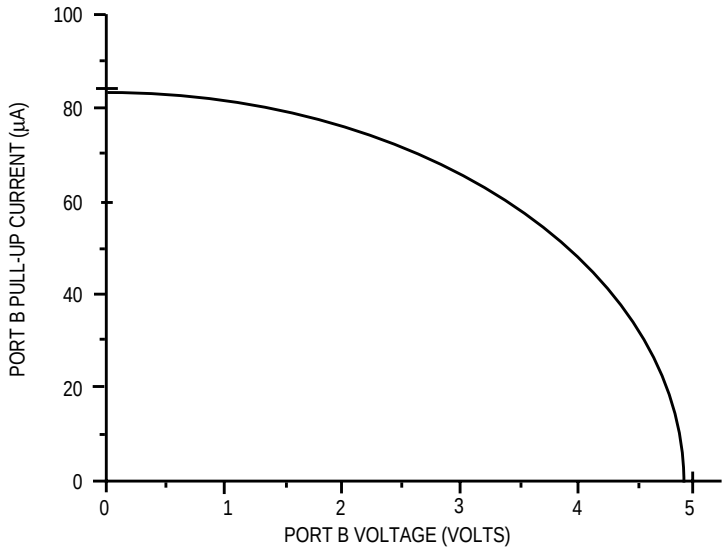

Figure 6. MC68HC705C8A Port B Pull-Up Current Source Characteristics V_{DD} = 5.0 V

68HC705C8A Key Pad Example Code

```
1         ************************************************
2         *                                              *
3         *         68HC705C8A key pad example code      *
4         *                                              *
5         ************************************************
6         0000  porta    equ   $00      * port A
7         0001  portb    equ   $01      * port B
8         0002  portc    equ   $02      * port C
9         0003  portd    equ   $03      * port D
10        0004  paddr    equ   $04      * port A  data direction register
11        0005  pbddr    equ   $05      * port B  data direction register
12        0006  pcddr    equ   $06      * port C  data direction register
13        0050  row      equ   $50      * row #
14        0051  col      equ   $51      * column #
15              *
16        ********************************
17        *                              *
18        *          Main Program        *
19        *                              *
20        ********************************
21  0200                org     $200
22  0200  A600 [2]      start   lda     #$00
23  0202  B700 [4]              sta     porta     * set port a low
24  0204  A655 [2]              lda     #$55
25  0206  B702 [4]              sta     portc     * set port c to $55
26  0208  A6FF [2]              lda     #$ff
27  020A  B704 [4]              sta     paddr     * port A - output
28  020C  B706 [4]              sta     pcddr     * port C - output
29  020E  A600 [2]              lda     #$00
30  0210  C71FDF[5]             sta     $1fdf     * make interrupt edge only
31                      *
32  0213  A60F [2]      lp1     lda     #$0f
33  0215  B701 [4]              sta     portb     * drive PB7-PB4 low
34  0217  43   [3]              coma
35  0218  B705 [4]              sta     pbddr     * make PB7-PB4 outputs
36  021A  9A   [2]              cli               * enable interrupts
37  021B  8E   [2]              stop              * enter stop mode
38  021C  20F5 [3]              bra     lp1       * reconfigure port B
39                      *
```

```
40        ***********************************
41        *                                 *
42        *    Interrupt Service Routine    *
43        *                                 *
44        ***********************************
45  1D00                org     $1d00
46  1D00  AD46  [6]     irqsev  bsr     bounce    * wait 30 ms for key debounce
47  1D02  B601  [3]             lda     portb
48  1D04  A10F  [2]             cmpa    #$0f      * check for a false interrupt
49  1D06  273F  [3]             beq     done
50
51  1D08  A610  [2]             lda     #$10      * start with the first column PB4
52  1D0A  B751  [4]             sta     col
53
54  1D0C  B651  [3]     scol    lda     col       * enable columns one at a time
55  1D0E  B705  [4]             sta     pbddr     * to determine the column
56
57  1D10  A610  [2]             lda     #$10      * wait until the pull-ups have
58  1D12  4A    [3]     lp3     deca              * had a chance to pull the
59  1D13  26FD  [3]             bne     lp3       * deselected columns high
60
61  1D15  A6FE  [2]             lda     #$fe      * check the rows one at a time
62  1D17  B750  [4]             sta     row
63
64  1D19  B601  [3]     scan    lda     portb     * read the rows
65  1D1B  AAF0  [2]             ora     #$f0      * don't care the high 4 bits
66  1D1D  B150  [3]             cmpa    row
67  1D1F  270F  [3]             beq     hold      * if match you found the row/
68                                                * col
69
70  1D21  B650  [3]             lda     row       * shift the row left and shift
71  1D23  43    [3]             coma              * in a "1"
72  1D24  48    [3]             lsla
73  1D25  43    [3]             coma
74  1D26  B750  [4]             sta     row       * save next row
75  1D28  A1EF  [2]             cmpa    #$ef      * check to see if any rows
76  1D2A  26ED  [3]             bne     scan      * left
77
78  1D2C  3851  [5]             lsl     col       * shift the column left
79  1D2E  24DC  [3]             bcc     scol
80
81
82  1D30  A6F0  [2]     hold    lda     #$f0      * wait here until the key
83  1D32  B705  [4]             sta     pbddr     * has been released
```

```
84  1D34  B601  [3]           lda     portb
85  1D36  A40F  [2]           anda    #$0f
86  1D38  A10F  [2]           cmpa    #$0f
87  1D3A  26F4  [3]           bne     hold
88
89  1D3C  3C00  [5]           inc     porta     * inc the # times a key has
90                                                * been pressed
91  1D3E  AD08  [6]           bsr     bounce    * wait for debounce
92
93  1D40  B651  [3]           lda     col       * write the row and column out
94  1D42  43    [3]           coma                * to port c
95  1D43  B450  [3]           anda    row
96  1D45  B702  [4]           sta     portc
97
98  1D47  80    [9]   done    rti
99
100
101 1D48  A627  [2]   bounce  lda     #$27      * debounce delay -
102 1D4A  AEFF  [2]   again   ldx     #$ff      *30 ms @ 2.0mhz
103 1D4C  5A    [3]   again2  decx
104 1D4D  26FD  [3]           bne     again2
105  1D4F  4A   [3]           deca
106  1D50  26F8 [3]           bne     again
107  1D52  81   [6]           rts
108
109
110  1FF0               org     $1ff0             * MOR1-enable PB3-PB0
111  1FF0  0F           fcb     $0f               * pullups
112
113  1FFA               org     $1ffa
114  1FFA  1D00         dw      irqsev            * IRQ vector
115  1FFE               org     $1ffe
116  1FFE  0200         dw      start             * reset vector

Errors: None
Labels: 20
Last Program Address: $1FFF
Last Storage Address: $0000
Program Bytes: $0076  118
Storage Bytes: $0000  0
```

General Rules for Using an EPROM MCU

The same rules for using EPROM MCUs apply to the 705C8A and the 705C8.

1. When using an EPROM, the window should be covered at all times except when UV erasing the units. Electrical tape works well to cover the window, but whatever is used should be completely opaque. When UV erasing the EPROM, the window should be clean of any residue from labels or tape used to cover the window — even gum from a label can prevent the EPROM from being erased.
2. The V_{PP} pin should be connected to V_{DD} at all times except when programming.