

AN499/D

Let the MC68HC705 program itself

By Åke Webjörn,
Motorola AB, Sweden

1 Introduction

There are several popular MCUs (Micro-Computer-Units) from Motorola on the market, which have their program memory stored in a built-in EPROM (Erasable-Programmable-Read-Only-Memory) or OTP (One-Time-Programmable) memory instead of the usual ROM (Read-Only-Memory). The difference between the EPROM and OTP part, is that there is a window in the package on the EPROM version which makes it possible to erase it under an UV-lamp for re-use. On the plastic OTP part, this window is missing, thus the EPROM array cannot be erased. OTPs are normally packaged in plastic which ensures a low cost high volume product.

EPROM memory cells require more mask layers in fabrication of the device, and testing of the EPROM cell is time consuming, which helps drive the cost higher than a normal ROM part. On delivery of the EPROM/OTP product another cost is the programming of the user program before the product is used. But it also means that the EPROM/OTP MCU becomes a more flexible product, allowing customer changes and requests to be met easily and enabling the product to be brought to market in a very short time. Some of the more popular types on the market are MC68HC711E9, MC68HC711D3, MC68HC705C8, MC68HC705P9 and MC68HC705J2.

The programming of the EPROM inside this kind of MCU is normally achieved with a built-in program that has been written and supplied by Motorola. This program is stored in a special area of ROM inside the MCU. The MCU starts to execute this built-in program under special circumstances, e.g., when the voltage on one or several pins at reset is above a certain level. This special mode of operation is called the bootloader mode. In this mode the MCU assumes that special programming hardware is connected to it. The bootloader then reads data from an external EPROM connected to the parallel I/O ports, or data from a serial port. Then it starts the programming by writing the data into the internal EPROM. It also controls the special programming voltage and keeps track of the time the cell inside the EPROM is programmed. In that way it provides a simple and efficient way for the customer to program the MCUs. Once completed, the MCU is inserted into its end application, and the user code in the EPROM is executed.

Sometimes it would be nice to be able to custom-program part or all of the built-in EPROM of the MCU, and to do so in the normal user mode rather than in the special bootstrap mode. The reason could be to be able to modify, add features or even remove parts of the standard program. Examples are adding program routines, storing serial numbers, calibration values, code keys, information on what external equipment is attached, removing test programs, etc.

1.1 Three examples of when this technique could be used

A traditional electronic door lock uses an MCU that compares the keys pushed, with a set of switches that determine the door opening code. If instead, the switches are stored in EPROM inside the MCU, then there is no way a burglar could open doors by simply breaking the lock cabinet, reading the switches and pushing the keys.

A second example is a small combustion engine. This needs a carefully adjusted air/gas mixture to minimise pollution. It is possible to write the program so that the MCU finds out the status of the engine and adapts to it. But this process may take a minute before the engine can give any output power; pollution will be quite large during this time. So it would be beneficial if the engine controller could memorise the last set-up values.

In a third example, a manufacturer wants to keep set-up/calibration routines for a system secret. With an EPROM-based MCU, it is simple to have a routine that, after the factory calibration or burn-in phase, simply removes the code by writing over the original set-up/calibration program with dummy code.

2 Contents of this application note

This application note is divided into three parts.

- The first part describes how the MCU is normally programmed in the special bootloader mode.
- The second part describes the design of hardware and software that allows the MCU to program itself.
- The third and last part gives some ideas on how to modify the program for enhancement.

The application note ends with the source code of the entire program.

3 How the programming is done

First let's look at how the built-in ROM program in an MC68HC705 MCU programs the EPROM inside.

3.1 Normal programming

Normally an MCU is run in the user mode. But to get access to the built-in ROM with the bootloader program, the MCU is powered up in a special way. This is done by resetting the MCU, that is by pulling the */RESET* line low, then keeping the */INT* at a high voltage while pulling the reset line high again.

See the Technical Data book for more information about the voltage required on the */INT* pin.

When the CPU (Central Processing Unit) inside the MCU, senses these conditions, it enters the special test mode. This mode makes the CPU start to fetch instructions from a small built-in ROM. The first thing that this program does is make the CPU read a port. The value on this port decides which program of the internal ROM should be run. Typical programs available are test routines used by the production and bootloader routines for programming and verifying the contents of the internal EPROM.

The programming routine in the bootloader program reads data from an external memory or from the serial port, and writes it into the EPROM. The verifying routine reads data from an external memory and compares it with the EPROM.

3.1.1 The program in more detail

Now let's look more closely at how a byte in the EPROM is programmed. The MC68HC705P9/D data book, section 11, is useful for reference.

The programming consists of the following steps:

- a. First the CPU sets the *latch* bit in the internal *eprog* register. This arms the EPROM data- and address bus latches.
- b. It then writes the data to the selected address in the EPROM array. Both data- and addressbus are latched.
- c. Using another port pin, the external programming voltage, *Vpp* is connected to the */INT-Vpp* pin.
- d. Then it sets the *epgm* bit in the *eprog* register. This connects *Vpp* power to the EPROM array.
- e. The program waits for the programming time which is 4 mS.
- f. Lastly, the *latch* and the *epgm* bits are cleared. This stops the programming and makes the EPROM behave as a normal memory again. The *Vpp* voltage is also removed.

In the bootloader mode the code to do this is fairly simple. To do it in user mode requires some extra effort. This is because the programming routine must be in a different memory space than the EPROM. When programming the EPROM cells, the CPU cannot execute instructions from the same memory area.

In user mode, the normal EPROM cannot be used to hold the programming software, because the address is latched with the value to be programmed. So the CPU cannot get its instructions from the EPROM, but must get them from elsewhere. The built-in ROM cannot be used either, because it is disabled in user mode. This means that the code must be put in the internal RAM (Random Access Memory).

The solution is to write a programming routine that is stored in EPROM. When the CPU wants to program the EPROM in user mode, it copies this routine out into RAM. It then calls the routine in RAM that does the programming. When complete, it returns to executing from EPROM.

The code of the programming routine is only 42 bytes long and the entire programming code takes 57 bytes. With the additional demonstration routines, the entire program is about 600 bytes.

4 The new approach

Now let's take a look at the new approach. First the hardware is discussed and then the software is described.

4.1 Hardware design

The test set-up is shown in [Figure 1](#). The board, called PRITSE for PROgram-IT-SELF, is shown to the left in the diagram. To the right it is connected with a serial cable to a PC or terminal.

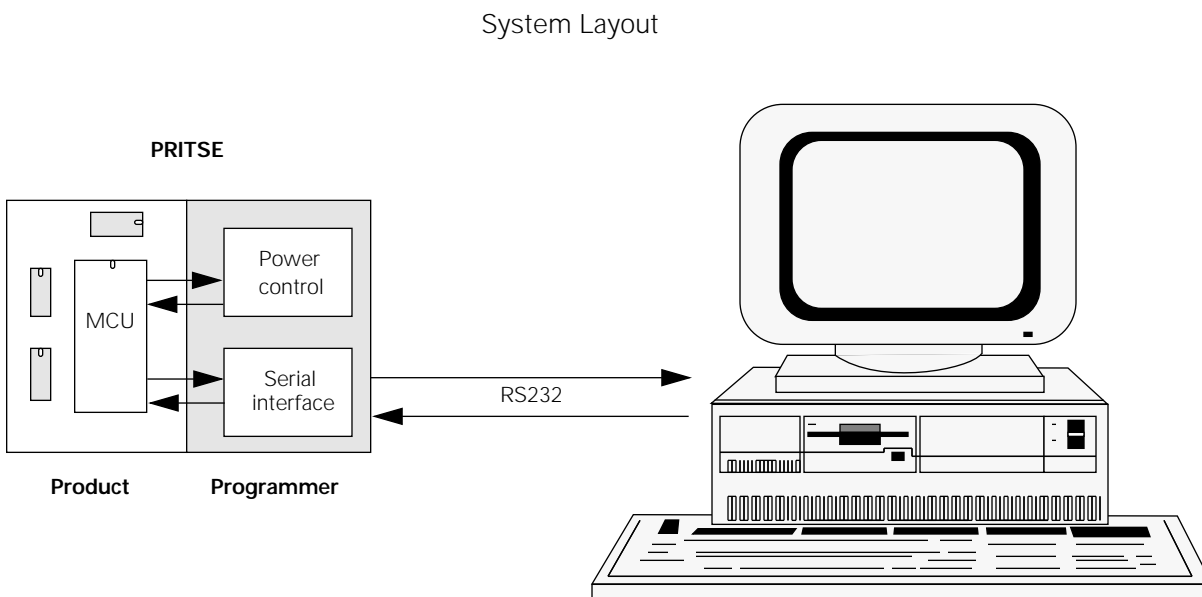


Figure 1. The test set-up

The PRITSE circuit board consists of two parts. In the white area the MCU and other components represent the finished product. The grey area, or programmer area, is what is added to program the MCU. The programmer area is connected for serial 9600 baud asynchronous communication with the outside world. On the other side it talks with the MCU with five I/O pins. The programmer area contains programming voltage control circuitry, an RS232 driver/receiver and one switch. This switch is used to select if normal operation or programming should take place.

Three different MCUs have been used to test the program. They are the MC68HC705P9, the MC68HC705P6 and the MC68HC705J2. For more information on these devices, see the technical reference manuals such as MC68HC705P9/D.

To be able to run the code on a large set of MCUs, no interrupts or complicated I/O port functions were used. For most designs it is sufficient to connect the MCU as the drawing shows and make some minor software adjustments.

A detailed schematic of the circuit board is shown in [Figure 2](#).

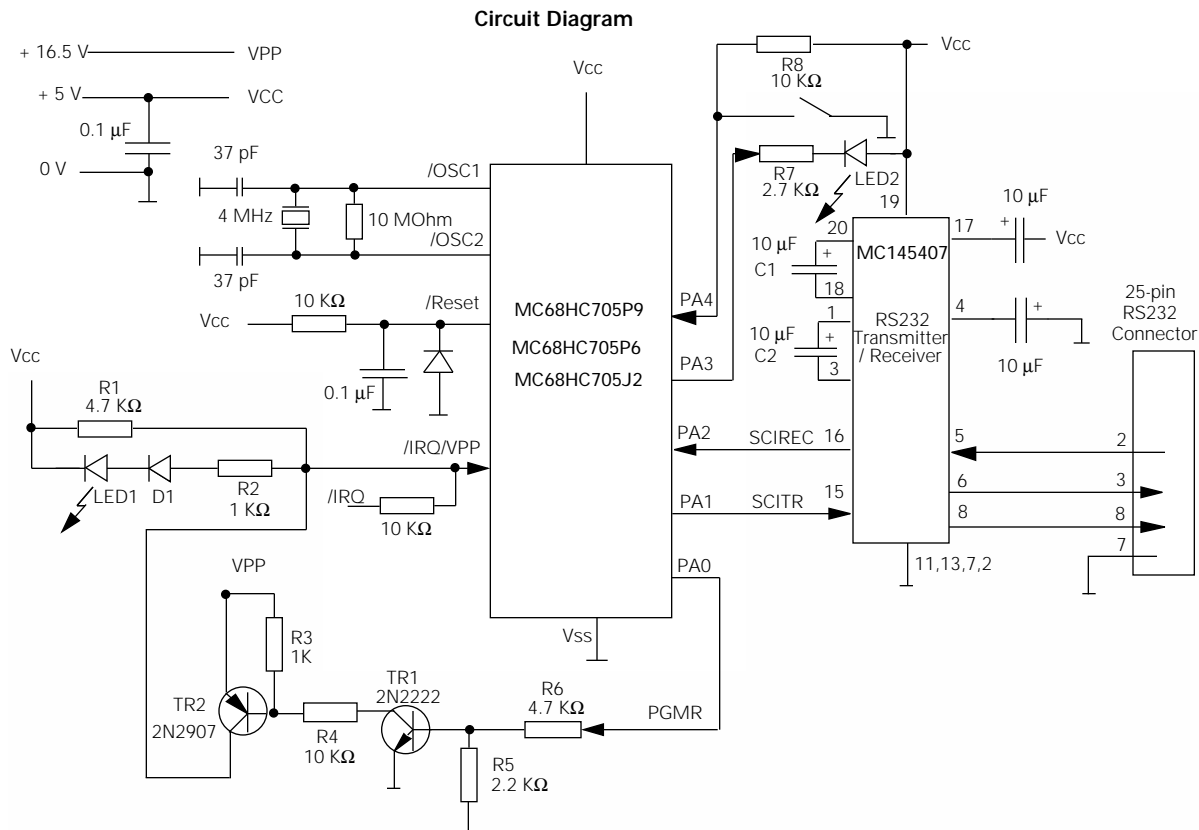


Figure 2. A schematic of the circuit board

Here follows a more detailed description of the hardware details of PRITSE.

4.1.1 Port PA0

This pin on the MCU, *PGMR* switches the high programming voltage to the */IRQ-Vpp* pin. The resistors *R5* and *R6* turn off the *Vpp* signal when *PA0* is in high-impedance state. This happens whenever the MCU is being reset. The resistor *R2*, the diode *D1* and the LED *LED1* are there for diagnostic purposes. The *LED1* turns on when */IRQ-VPP* is higher than *Vcc*.

4.1.2 Port A1

PA1 is the serial SCI transmitter. It is implemented in software and runs in half duplex mode. The standard speed is 9600 baud, but can easily be changed in software.

4.1.3 Port A2

PA2 is the serial SCI receiver, also implemented in software. It runs in half duplex at the same speed as the transmitter.

4.1.4 Port A3

PA3 is connected to an LED2 for diagnostic purpose. In the program it is set to turn on while the CPU fetches instructions from the RAM.

4.1.5 Port A4

PA4 is an input from a switch. It is used to select between normal operation or programming mode. See paragraph 4.3 for further details.

4.2 Software implementation

The software is written to be easy to understand. It is divided into five modules (see [Figure 3](#)):

<i>MCUTYPE.ASM</i>	MCU type declarations
<i>MACRO.ASM</i>	Macro routines
<i>UTILITY.ASM</i>	General utility programs
<i>PROG.ASM</i>	Reading and writing from the EPROM
<i>PRITSE.ASM</i>	Main program

The modules are not linked to each other but assembled as one big file. There is one large module called *PRITSE.ASM*. All the other programs are included in this module.

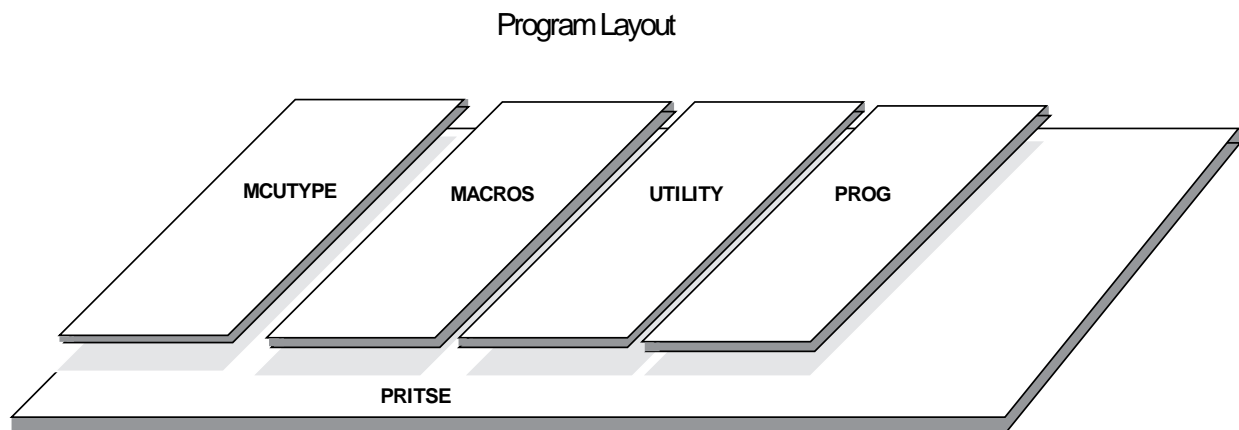


Figure 3. The relationship of each program module

The *MCUTYPE.ASM* describes the address map of the selected MCU. The MC68HC705P9 memory map is displayed. In appendix 1 and 2, the MC68HC705J2 and the MC68HC705P6 are shown. If another MCU is used, it is simple to change the contents to *MCUTYPE.ASM*.

The *MACRO.ASM* contains a set of simple macros for handling in- and outports, messages, and conditional jumps. The purpose of the macros is to make the source code easier to understand. This program was written with P&E IASM05 macro assembler. It may be necessary to change a few macros, if another assembler is to be used.

The *UTILITY.ASM* contains a number of subroutines. They are used by the debugging part of the program. Most of the routines make interfacing with a standard terminal easy. They can therefore be of interest in other applications.

PROG.ASM contains the routines for reading and programming the EPROM. These are the routines that are of major importance to this application.

PRITSE.ASM is the main program. As mentioned before, the other programs are not linked, but handled by the assembler IASM05 as include files.

A condition called *debug* is set or cleared in the beginning of *PRITSE.ASM*. When set, this condition turns off the code that turns on the actual programming voltage. The *debug* condition is needed when debugging the program with an emulator.

4.3 Software design

The MCU can run in two ways. The flow of the program is shown in [Figure 4](#).

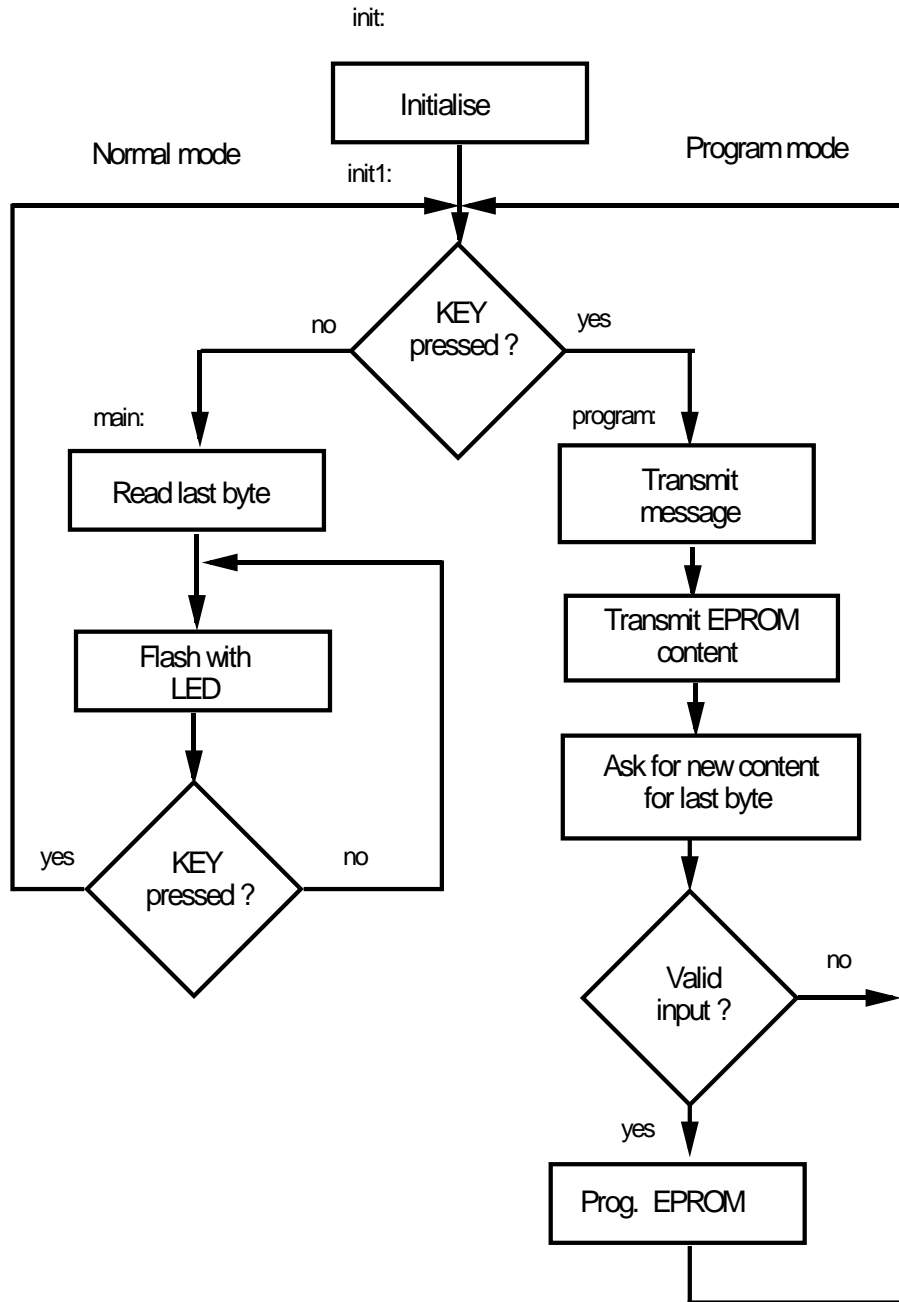


Figure 4. The program flow

If normal mode is selected with pin *PA4* set on the MCU, it flashes with the *LED2* connected to *PA3*. The speed of the flashing is proportional to the last programmed byte in the EPROM. The delay is done by decrementing a timer loaded with the data of the last byte written into the EPROM.

If programming mode is selected instead, it will behave as follows:

- a. Transmits a message to the terminal, telling which version its internal program has.
- b. It displays the EPROM buffer that is 256 bytes long by writing its hexadecimal values on the external PC or terminal.
- c. Then it asks the user for a new value to program.
- d. If a valid key combination is entered, the program continues, else it loops back to b.
- e. The EPROM is programmed and the program loops back to b.

The complete program list is shown as list 1 in Appendix 3.

The description that follows covers only the programming procedures.

4.3.1 Programming model

Figure 5 gives a short description of what the address range of the MC68HC705 MCU looks like. The I/O ports are at the low addresses. Then a bit higher up comes the internal RAM. This is used both for storage of variables and for the stack. And even higher up comes the EPROM, which is used for storing the program. Three labels are shown. *Prog_eprom* and *prog_rout* are the routines that do the programming. A third label, *EPROM_area*, is shown at a higher address. This label points at the area which is free for writing variables.

4.3.2 The *prog_eprom* routine

When programming is needed, the *prog_eprom* routine is called. See Figure 6.

- a. It starts by looking for a free EPROM byte that has not been programmed before. It begins at the address *EPROM_area + 255* and scans downward. If a free byte is found before the pointer passes *EPROM_area*, the program continues.
- b. The next step is to copy the routine *prog_rout* to the RAM. The start address is called *RAM_area*.
- c. Then it gets the byte to store from the cell *eprom_data*, which in this example is \$9B. It has been stored there by the software SCI.
- d. The CPU then jumps to *prog_rout* that now can be found in *RAM_area*

4.3.3 The *prog_rout* routine

The program continues to run at the *RAM_area* label (see Figure 7).

- a. First it sets the *pgmr* bit in *porta*. This turns on the programming voltage to the MCU.
- b. Then self modifying code is used to modify the address at *selfmod*. This is a full 16-bit address used by the 'STA' instruction.
- c. The *latch* bit is set in *eprog*. Now the EPROM is waiting for code input and is no longer available for execution.
- d. The code that is in *eprom_data* is copied to the modified address which is stored at *selfmod*.
- e. The *epgm* bit is set in *eprog*. This starts the programming. Then it waits for 4 mS while the EPROM is programmed.
- f. The *latch* and *epgm* bits in *eprog* register are cleared. This stops programming and enables the EPROM for normal execution again.
- g. Finally, the *pgmr* bit in *porta* is cleared to remove the high programming voltage and to return to *prog_eprom*.

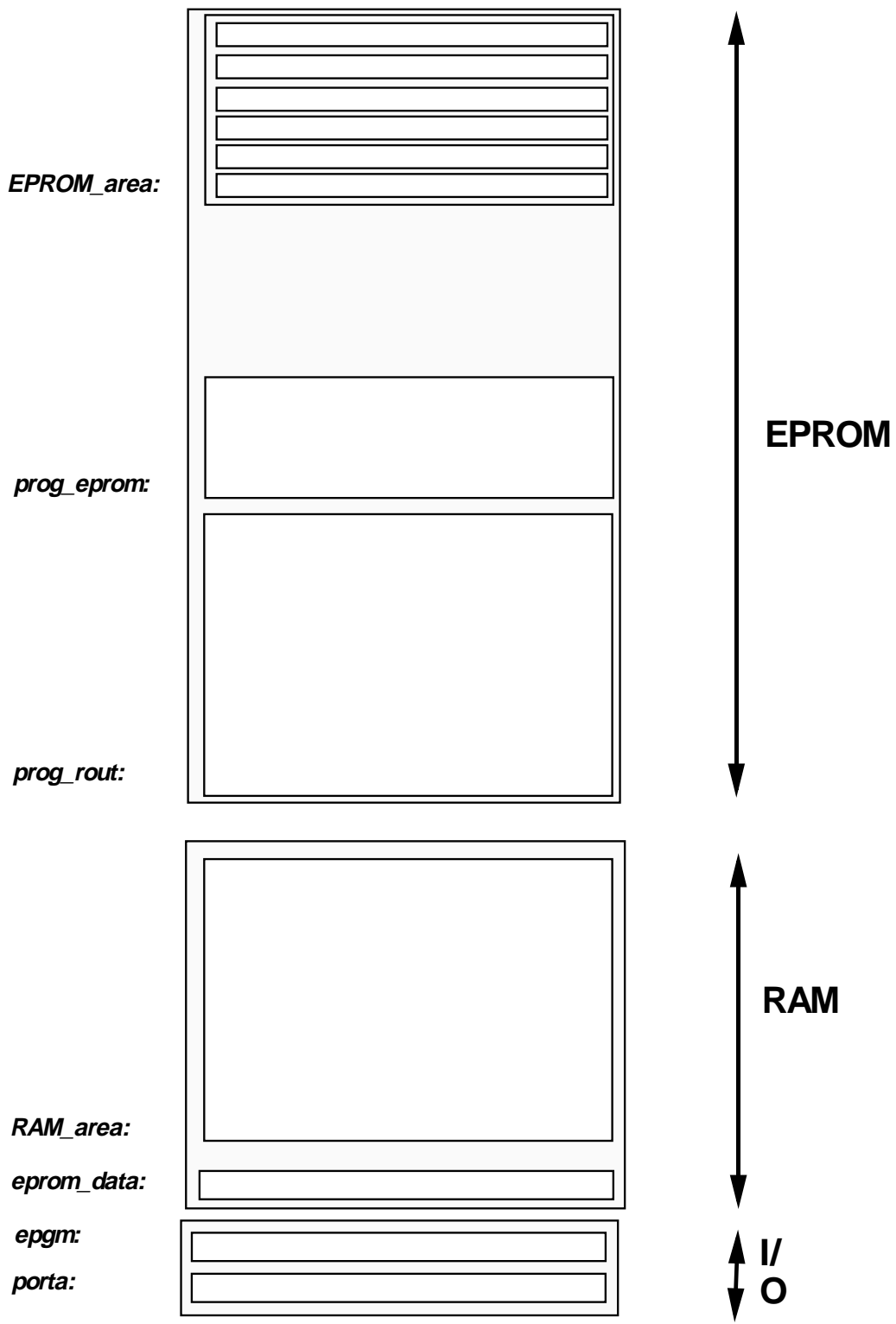


Figure 5. The address range of the MCU

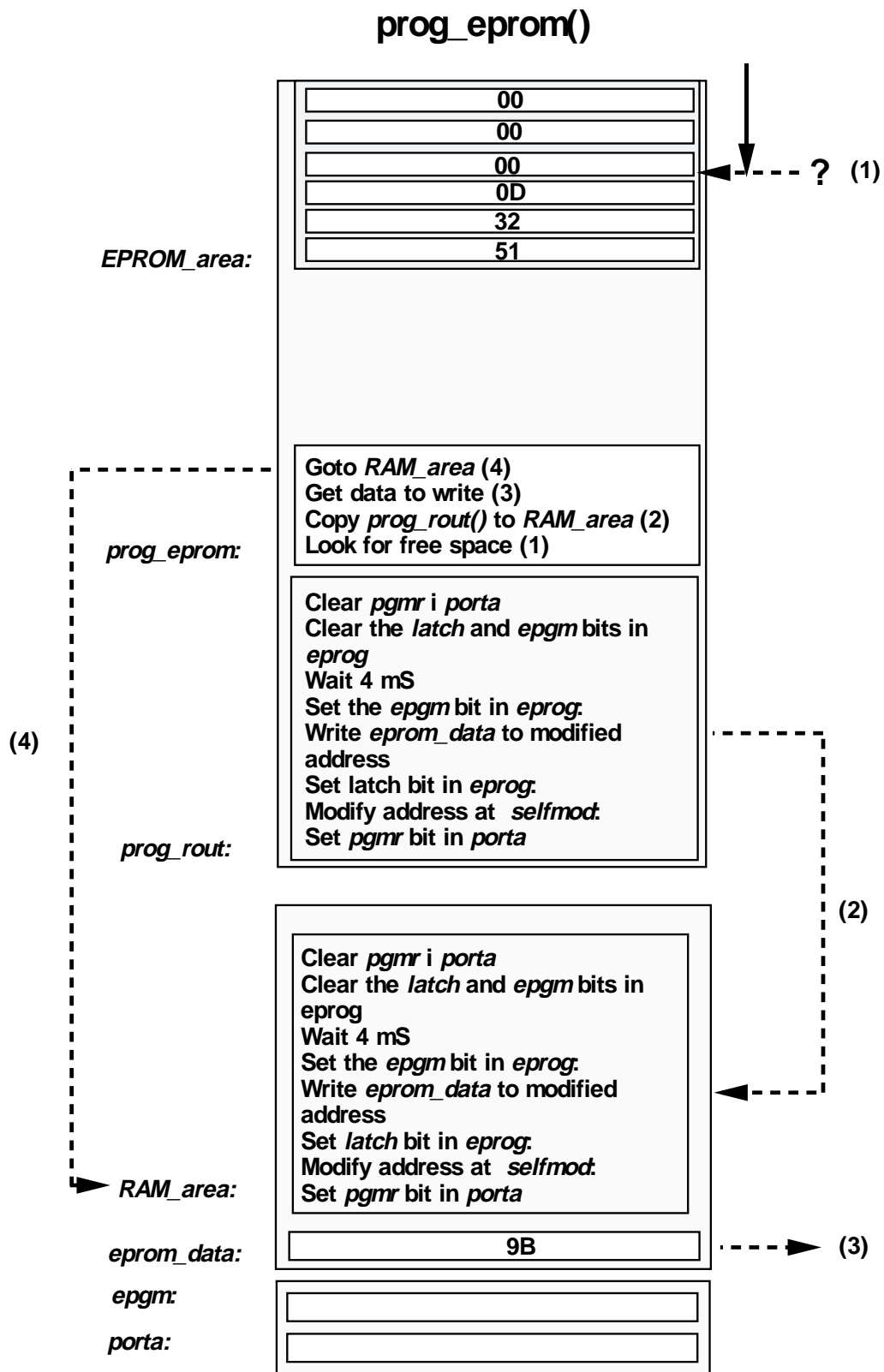


Figure 6. The prog_eprom routine

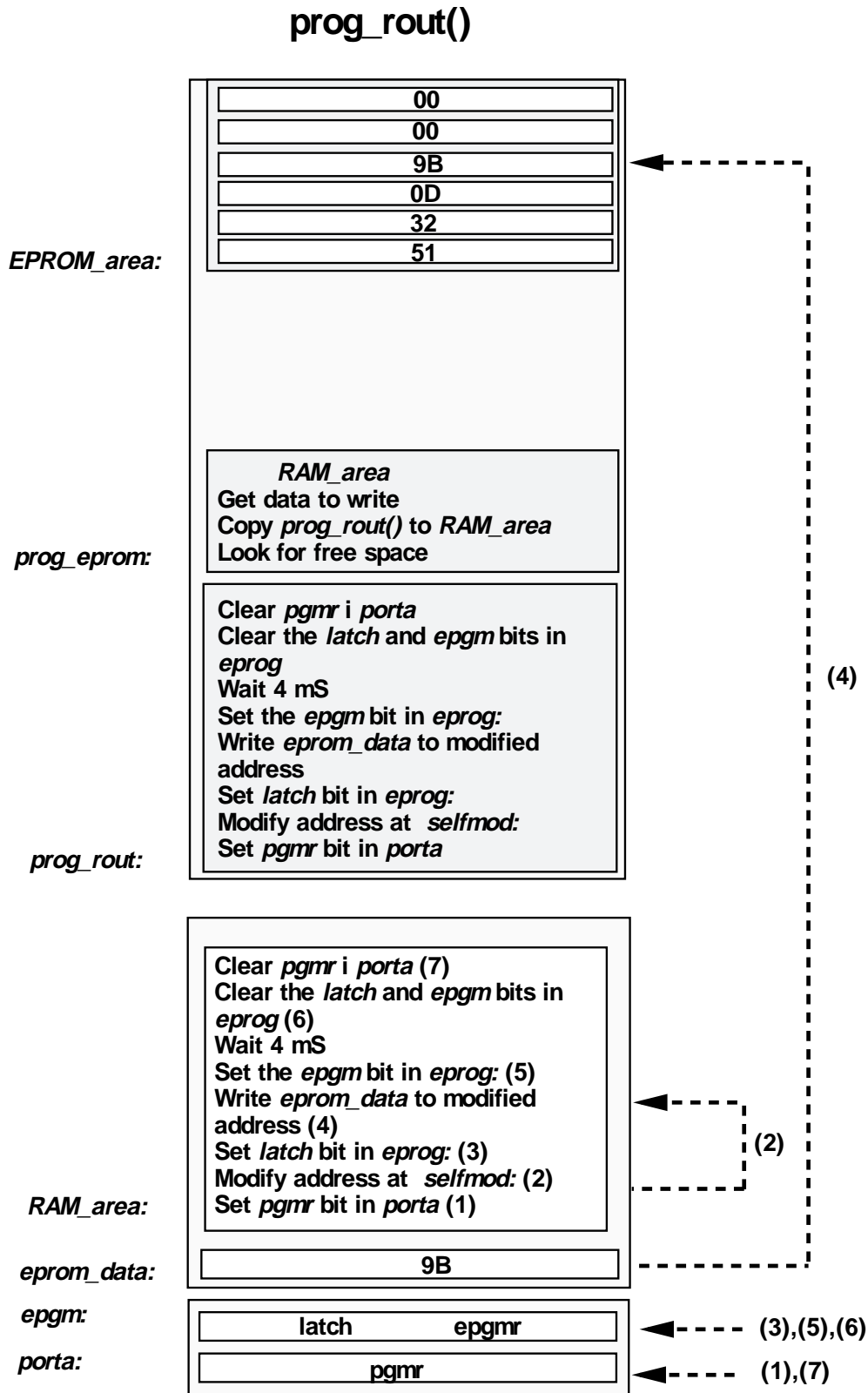


Figure 7. The prog_rout routine

5 Suggested improvements

Here are some ideas for improvements to the standard software.

5.1 To remove a program

There can be parts of the program that should be removed before leaving the factory. However an MCU with EPROM cells cannot be partially erased.

A way of making bytes in the EPROM unreadable is to program all bits, that is, to write '\$0FF' in the cells. Now '\$0FF' is interpreted by the MC68HC05 processors as the instruction 'STX ,X'.

This means that before calling a routine that might be erased, the X register should point at a harmless location in the first 256 bytes of the memory map. The routine should of course be terminated with a 'RTS' instruction.

Here is an example of this code where the routine *calib* has been removed.

```
        LDX    #stack_bottom    ;point at harmless location
        JSR    calib            ;call the procedure
        RTS

calib:  STX    ,X                ;the original code is removed
        STX    ,X
        RTS                    ;until the last RTS instruction
```

5.2 To handle larger programs

To modify the code so that it can handle programs larger than 256 bytes is quite easy. The routine *find_free()* must be changed to handle the larger address range.

Note that the routine *read()* is made too complicated. There is no need to jump out into the RAM, just to read a byte of EPROM. The reason that this routine was made so unnecessarily complicated was to make it easy to handle larger programs than 256 bytes.

5.3 Download the programming algorithm

It is of course possible to not include any programming algorithm at all in the program, and still do programming. What is required is a driver, e.g., for a serial port. The code, *prog_rout*, which is about 60 bytes, is downloaded together with the data and address to the RAM. The program then programs the data into the EPROM cells, and disappears when the power is removed. This gives most flexibility.

6 Conclusion

This application note shows that it is quite simple to add EPROM programming to MC68HC705 microcomputer applications. I hope that it suggests some new ideas on how to tackle and solve the EPROM programming problem.

Acknowledgements

The author acknowledges the help and assistance of his colleagues Jeff Wright, Dugald Campbell and Anthony Breslin.

Appendix 1

```
.PAGE
.SUBHEADER 'MCU type'
;last change 94-02-18
;=====
;=          MC68HC705J2      =
;=====
erased      EQU    $0      ;erase EPROM-cell

                ORG    $0
porta       RMB    1        ;port A
DDR         EQU    4        ;offset for data direction reg.
pgmr        EQU    0        ;to turn on programming voltage
scitr       EQU    1        ;SCI transmitter
scirec      EQU    2        ;SCI receive register
LED         EQU    3        ;to drive diagnostic LED

; EPROM programming register
                ORG    $1C
eprog       RMB    1
epgm        EQU    0        ;bit 0
latch       EQU    2        ;bit 2
pr_time     EQU    4        ;time in mS

; memory parameters
ram_start   EQU    $90
rom_start   EQU    $700
rom_end     EQU    $F00

; Mask Option Register
mor_adr     EQU    $F00
mor         EQU    $0

;Reset vector
reset_vector EQU $FFE
```

Appendix 2

```
.PAGE
.SUBHEADER 'MCU type'
;last change 94-03-04
;=====
;=          MC68HC705P6      =
;=====
erased      EQU    $0      ;erase EPROM-cell

          ORG    $0

porta      RMB    1      ;port A
DDR        EQU    4      ;offset for data direction reg.
pgmr       EQU    0      ;to turn on programming voltage
scitr      EQU    1      ;SCI transmitter
scirec     EQU    2      ;SCI receive register
LED        EQU    3      ;to drive diagnostic LED

; EPROM programming register
          ORG    $1C
eprog      RMB    1
epgm       EQU    0      ;bit 0
latch      EQU    2      ;bit 2
pr_time    EQU    4      ;time in mS

; memory parameters
ram_start  EQU    $50
rom_start  EQU    $100
rom_end    EQU    $1300

; Mask Option Register
mor_adr    EQU    $1F00
mor        EQU    $0

;Reset vector
reset_vector EQU $1FFE
```

Appendix 3

PRITSE.ASM
PROgram-IT-SElf

Assembled with IASM 03/10/1994 10:13 PAGE 1

```
1 ; last change 94-03-09
2
3 ;*****
4 ;*          PROgram-IT-SElf          *
5 ;*****
6 ; This program shows how the MCU
7 ;   programs its own EPROM
8
0000 9 $BASE 10T
0000 10 ; $SET debug ;determines if debug mode
0000 11 $SETNOT debug
```

RITSE.ASM
PROgram-IT-SElf
Main program

Assembled with IASM 03/10/1994 10:13 PAGE 2

```
0000 11 $INCLUDE "pritse\mcutype.asm"
```

RITSE.ASM
PROgram-IT-SElf
MCU type

Assembled with IASM 03/10/1994 10:13 PAGE 3

```
11 ;last change 94-03-09
12 ;=====
13 ;=          MC68HC705P9          =
14 ;=====
0000 15 erased          EQU    $0          ;erase EPROM-cell
16
0000 17 ;Ports on the MCU
0000 18          ORG    $0
0000 19 porta          RMB    1          ;port A
0001 20 DDR            EQU    4          ;offset for data
21          ;direction reg.
0001 22 pgmr           EQU    0          ;to turn on programming
23          ;voltage
0001 24 scitr          EQU    1          ;SCI transmitter
0001 25 scirec         EQU    2          ;SCI receive register
0001 26 LED            EQU    3          ;to drive diagnostic LED
0001 27 key            EQU    4          ;key to switch modes
28
29 ; EPROM programming register
001C 30          ORG    $1C
001C 31 eprog          RMB    1
001D 32 epgm           EQU    0          ;bit 0
001D 33 latch         EQU    2          ;bit 2
001D 34 pr_time       EQU    4          ;time in mS
35
36 ; memory parameters
001D 37 ram_start      EQU    $80
001D 38 rom_start      EQU    $100
001D 39 rom_end        EQU    $900
40
41 ; Mask Option Register
001D 42 mor_adr        EQU    $900
001D 43 mor            EQU    $0
44
45 ;Reset vector
001D 46 reset_vector   EQU    $1FFE
47
001D 48 $INCLUDE "pritse\macros.asm"
```

```
48 ; last change 94-03-09
49 ;=====
50 ;= Macros for the assembler routine =
51 ;=====
52
001D 53 $MACRO inport
54         BCLR    %1,{%2+DDR}
001D 55 $MACROEND
56
001D 57 $MACRO outport
58         BSET    %1,{%2+DDR}
001D 59 $MACROEND
60
001D 61 $MACRO message
62         LDX     #{%1-msg}
63         JSR     xmitmsg
001D 64 $MACROEND
65
001D 66 $MACRO if_smaller
67         CMPA    #{%1}
68         BCS     %2
001D 69 $MACROEND
70
001D 71 $MACRO if_larger
72         CMP     #{%1+1}
73         BCC     %2
001D 74 $MACROEND
75
001D 76 $MACRO if_equal
77         CMP     #{%1}
78         BEQ     %2
001D 79 $MACROEND
80
001D 81 $MACRO if_not_equal
82         CMP     #{%1}
83         BNE     %2
001D 84 $MACROEND
85
001D 86 $INCLUDE "pritse\utility.asm"
```



```
86 ; Last change 94-03-10
87
88 ;=====
89 ;=          Utility Routines          =
90 ;=====
91
92 ;=====
93 ;=          Symbolic absolute values   =
94 ;=====
95
001D 96 del24          EQU    134T    ; bitwait for 1200 baud,
97                                     ; @ 4 Mhz
001D 98 stopbit       EQU    2        ; two stop bits
001D 99 cr           EQU    $0DH    ; carriage return
001D 100 lf        EQU    $0AH    ; line feed
001D 101 esc       EQU    $1BH    ; escape
001D 102 bell      EQU    $07H    ; bell
103
104 ;=====
105 ;=          Start of RAM area         =
106 ;=====
107
0080 108                ORG    ram_start
109
110 ; SCI data
0080 111 bitcount    RMB    1        ; bit counter for transmit
0081 112 tr_char     RMB    1        ; tmp storage for transmit
0082 113 rec_char    RMB    1        ; tmp storage for transmit
0083 114 sav_char    RMB    1        ; tmp storage for transmit
0084 115 hex        RMB    1        ; tmp storage for tohex
0085 116 strptr      RMB    1        ; string pointer
117
118 ; display data
0086 119 bytecount   RMB    1        ; byte counter
0087 120 colcount    RMB    1        ; column counter
0088 121 count      RMB    1        ; counter
0089 122 src_adr    RMB    1        ; source address
008A 123 dst_adr    RMB    1        ; destination address
124
125 ; eprom data
008B 126 eprom_data  RMB    1        ; data to eprom
008C 127 adr_hi     RMB    1        ; address, high byte
008D 128 adr_lo     RMB    1        ; address, low byte
129
130 ;=====
131 ;=          Start of free RAM area     =
132 ;=====
133 ;Here starts empty RAM area used by relocated programs
008E 134 RAM_area:    ORG    $
135
```

```

135 ;=====
136 ;=          Start of ROM area          =
137 ;=====
0100 138          ORG          rom_start
139
140 ;=====
141 ;=          Bitwait delay routines     =
142 ;=====
143 ; Function: bitwait(,)
144 ; Description: Delay for asynchronous transmission
145 ; Input: delay in reg A
146 ; Output: none
147 ; Uses: none
148 ; Note:   bitwait formula: bitwait = 32 + 6 ; A cycles
149 ; bit time for 9600 baud is 104 uS or 208 cycles at 4 Mhz
150 ; A = 30 gives a bit time of 106 uS, or an error of < 2%
151 ; minimum baudrate is about 1300 baud
152 ; Reg X is not used.
153 halfbitwait:
0100 A643 154          LDA          #{del24 / 2}      ;2 cycles
0102 2002 155          BRA          bitwait1         ;3 cycles
156
157 bitwait:
0104 A686 158          LDA          #del24            ;2 cycles
159
160 bitwait1
0106 4A   161          DECA                     ;3 cycles
0107 26FD 162          BNE          bitwait1         ;3 cycles
0109 81   163          RTS                     ;6 cycles
164
165
166 ;=====
167 ;=          Transmit one character      =
168 ;=====
169 ; Function: transmit(a,)
170 ; Description: Transmit one character
171 ; Input: character to transmit in reg A
172 ; Uses: char, bitcounter
173 ; Output: none
174 ; Uses: tr_char, bitcount, porta
175 ; Note:
176 transmit:
010A B781 177          STA          tr_char           ;save the char in rot. buffer
010C A609 178          LDA          #9              ;prepare to transmit 9 bits
010E B780 179          STA          bitcount         ;save it in bitcount
0110 1200 180          BSET        scitr,porta      ;pull scitr high
0112 ADF0 181          BSR          bitwait         ;wait one bit time
0114 1300 182          BCLR        scitr,porta      ;send start bit
183
184 ; transmit one bit
185 tra3:
0116 ADEC 186          BSR          bitwait         ; 6 cycles
0118 3A80 187          DEC          bitcount        ; 5 cycles
011A 270C 188          BEQ          tra2            ; 3 cycles
011C 3681 189          ROR          tr_char         ; 5 cycles
011E 2504 190          BCS          tral           ; 3 cycles
191          ;-----

```

```

192                                     ; 32 (see bitwait routine)
0120 1300          193          BCLR   scitr,porta  ;send 0
0122 20F2          194          BRA    tra3
195
196  tra1:
0124 1200          197          BSET   scitr,porta  ;or send 1
0126 20EE          198          BRA    tra3
199
200  tra2:
0128 1200          201          BSET   scitr,porta  ;send stop bit
012A ADD8          202          BSR    bitwait   ;wait one period
012C 81            203          RTS
204
205
206  ;=====
207  ;=          Transmit ROM message          =
208  ;=====
209  ; Function: xmit_msg(,x)
210  ; Description: Transmit message stored in ROM
211  ; Input: X contains offset in msg area
212  ; Uses:  strptr
213  ; Output: none
214  ; Uses:  strpptr
215  ; Note: This routine is called by the macro 'message'
216  ; The message is terminated with 0
217
218  msg:                                     ;relativ address
219  init_msg:
012D 0D0A2020     220          DB      cr,lf,'  Program-IT-Self, V 1.0',cr,lf,0
      20202050
      526F6772
      616D2D49
      542D5345
      6C662C20
      5620312E
      300D0A00
221  buffer_msg:
014D 0D0A0D0A     222          DB      cr,lf,cr,lf,'Buffer content:',cr,lf,0
      42756666
      65722063
      6F6E7465
      6E743A0D
      0A00
223  quest_msg:
0163 203F20     224          DB      ' ? '
225  nl_msg:
0166 0D0A00     226          DB      cr,lf,0
227  data_msg:
0169 0D0A4461     228          DB      cr,lf,'Data: ',0
      74613A20
      00
229  mem_full_msg:
0172 0D0A4D65     230          DB      cr,lf,'Memory full',0
      6D6F7279
      2066756C
      6C00
231

```

```

232 xmitmsg:
0180 BF85 233     STX     strptr      ; save pointer in strptr
234
235 xmitmsg2:
0182 BE85 236     LDX     strptr      ; get pointer to X
0184 D6012D 237     LDA     msg,X       ; get character
0187 2707 238     BEQ     xmitmsg1    ; done if 0
0189 CD010A 239     JSR     transmit   ; else send one character
018C 3C85 240     INC     strptr      ; move pointer
018E 20F2 241     BRA     xmitmsg2
242
243 xmitmsg1:
0190 81 244     RTS              ; return back
245
246 ;=====
247 ;=          Convert to hexadecimal          =
248 ;=====
249 ; Function: to_ascii(a)
250 ; Description: Transmits byte as a 2 digit hexadecimal value
251 ; Input: A contains byte to convert
252 ; Output: none
253 ; Uses: hex, hexstr
254 hexstr:
0191 30313233 255     DB      '0123456789ABCDEF'
      34353637
      38394142
      43444546
256
257 to_ascii:
01A1 B784 258     STA     hex        ; save hex value
01A3 44 259     LSRA                ; shift right 4 times to
260                               ; get high nibble
01A4 44 261     LSRA
01A5 44 262     LSRA
01A6 44 263     LSRA
01A7 97 264     TAX              ; put result in x
01A8 D60191 265     LDA     hexstr,X   ; translate to ASCII
01AB CD010A 266     JSR     transmit   ; transmit result
267
01AE B684 268     LDA     hex        ; get hex value again
01B0 A40F 269     AND     #$F       ; mask low nibble
01B2 97 270     TAX
01B3 D60191 271     LDA     hexstr,X   ; translate to ASCII
01B6 CD010A 272     JSR     transmit   ; transmit low nibble
273
01B9 A620 274     LDA     #' '
01BB CD010A 275     JSR     transmit   ; finish with a space
01BE 81 276     RTS
277
278 ;=====
279 ;=          Convert from ASCII to hexadecimal  =
280 ;=====
281 ; Function: to_hex(a): byte,carry flag
282 ; Description: Translates a byte as a 2 digit
283 ; hexadecimal value
284 ; Input: A contains byte to convert
285 ; Output: return value in A. Carry flag if bad input

```

```

286 ; Uses: none
287 to_hex:
01BF macro 288 if_smaller #'a',to_hex2
01C3 A020 289 SUB #20 ;convert lower case
290 to_hex2:
01C5 macro 291 if_larger #'F',to_hex5 ;if > 'F' jump
01C9 macro 292 if_smaller #'0',to_hex5 ;if < '0' jump
01CD macro 293 if_larger #'@',to_hex1 ;if > '@' jump
01D1 macro 294 if_larger #'9',to_hex5 ;if > '9' jump
295
296 to_hex1:
01D5 A030 297 SUB #'0' ;convert to decimal
01D7 macro 298 if_smaller #10T,to_hex3
01DB A007 299 SUB #{'A'-'9'- 1}
01DD 2502 300 BCS to_hex5
301 to_hex3:
01DF 98 302 CLC ;no errors, clear carry
01E0 81 303 RTS ;and return
304
305 to_hex5:
01E1 99 306 SEC ;error, set carry
01E2 81 307 RTS ;and return
308
309
310 ;=====
311 ;= Receive one character =
312 ;=====
313 ; Function: receive(): byte,carry flag
314 ; Description: Receveives one character
315 ; Input: none
316 ; Output: character that is received in reg A
317 ; Uses: rec_char, porta
318 ; Note:
319 receive:
01E3 3F82 320 CLR rec_char ;clear rec_char
01E5 AE08 321 LDX #8 ;load 8 in index
322
323 rec0:
01E7 0500FD 324 BRCLR scirec,porta,rec0 ;wait for idle line
325
326 rec1:
01EA 0400FD 327 BRSET scirec,porta,rec1 ;wait for start bi
01ED CD0100 328 JSR halfbitwait ;wait 1/2 bit
329
330 rec2:
01F0 CD0104 331 JSR bitwait ;wait 1 bit
01F3 B682 332 LDA rec_char ;read the rec_char
01F5 44 333 LSRA ;shift right
01F6 050002 334 BRCLR scirec,porta,rec3 ;if bit is 0, jump
01F9 AA80 335 ORA #80 ;else add 1
336 rec3:
01FB B782 337 STA rec_char ;save result
01FD 5A 338 DECX ;decrement bit count
01FE 26F0 339 BNE rec2 ;any bits left ?
0200 81 340 RTS ;if no, the return
341
342

```

```
343
344 ;=====
345 ;=          Receive one byte          =
346 ;=====
347 ; Function: recbyte(): byte,carry flag
348 ; Description: Receives two characters as one byte
349 ; Input:     none
350 ; Output:    byte A, carry flag set if bad input.
351 ; Uses:     hex, rec_char
352 ; Note:
353 recbyte:
0201 3F84 354     CLR     hex           ;clear result
355
356 recb1:
0203 CD01E3 357     JSR     receive       ;get a character
0206 B782 358     STA     rec_char      ;save the char
0208 CD010A 359     JSR     transmit      ;echo the result back
020B B682 360     LDA     rec_char      ;get rec_char
020D macro 361     if_equal    #cr,recb2
0211 ADAC 362     BSR     to_hex       ;convert to hex
0213 2512 363     BCS     recb3      ;done if no character
0215 3884 364     LSL     hex         ;shift result left 4 times
0217 3884 365     LSL     hex
0219 3884 366     LSL     hex
021B 3884 367     LSL     hex
021D BA84 368     ORA     hex         ;add new value
021F B784 369     STA     hex         ;save it
0221 20E0 370     BRA     recb1     ;jump back to start again
371
372 recb2:
0223 B684 373     LDA     hex         ;return result
0225 98 374     CLC           ;with no carry
0226 81 375     RTS
376
377 recb3:
0227 A607 378     LDA     #bell
0229 CD010A 379     JSR     transmit      ;transmit a bell char.
022C 99 380     SEC           ;set error flag
022D 81 381     RTS           ;and return
382
022E 383     $INCLUDE "pritse\prog.asm"
```

```

383
384 ; last change 94-03-09
385 ;=====
386 ;=          Reading and writing to          =
387 ;=          and from the EPROM            =
388 ;=====
389
390 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
391 ;x          Read routine                    x
392 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
393 ; Function: read_rout(,)
394 ; Input: address in adr_hi and adr_lo
395 ;       data in eprom_data
396 ; Output: none
397 ; Uses: adr_hi, adr_lo, RAM_area
398 ; Note: this program should be loaded at RAM address
399 ; 'RAM_area' and uses selfmodifying (!!) code.
400 ; This routine is unnecessary complex as the read operation
401 ; can be made in a much simpler way.
402 read_rout:
403
404 ;modify address
022E B68C 405     LDA     adr_hi
406
407 ;high byte
0230 C70099 408     STA     {RAM_area+readr1+1-read_rout}
409
410 ;low byte
0233 B68D 411     LDA     adr_lo
0235 C7009A 412     STA     {RAM_area+readr1+2-read_rout}
413
414 ;next instruction is modified by the program itself
415 readr1:
0238 C6FFFF 416     LDA     $0FFFF           ;read the data
023B 81     417     RTS
418
419 read_end:
023C      421     read_size     EQU     {read_end - read_rout}
422
423
424 ;=====
425 ;=          Read          =
426 ;=====
427 ; Function: read(,)
428 ; Description: copy the data from EPROM to RAM
429 ; starting with the label "RAM_area"
430 ; then jumps into this routine.
431 ; Input: address in adr_hi and adr_lo
432 ; Output: data in reg A
433 ; Uses: read_rout, RAM_area
434 read:
023C AE0E 435     LDX     #read_size       ;no of bytes to relocate
436
437 read1:
023E D6022D 438     LDA     {read_rout-1},X ;get data source
0241 E78D 439     STA     {RAM_area-1},X ;store in dest

```

```

0243 5A      440      DECX
0244 26F8    441      BNE      read1      ;loop until routine is copied
                                442      ;into RAM
                                443
0246 BD8E    444      JSR      RAM_area    ;call the routine in RAM
                                445
0248 81      446      read2:
                                447      RTS
                                448
                                449
                                450      ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                                451      ;x          Programme routine          x
                                452      ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                                453      ; Function: prog_rout(,)
                                454      ; Description: This is the central programming routine
                                455      ; Input: address in adr_hi and adr_lo
                                456      ;      data in eprom_data
                                457      ; Output: none
                                458      ; Uses: porta, adr_hi, adr_lo, eprom_data, eprog
                                459      ; Note this program should be loaded
                                460      ; at address 'RAM_area' and uses selfmodifying (!! ) code.
                                461
                                462      prog_rout:
0249      463      $IFNOT debug
0249 1000     464      BSET      pgmr,porta    ;turn on VPP
024B macro   465      outport pgmr,porta
024D      466      $ENDIF
                                467
024D B68C     468      LDA      adr_hi      ;modify address
024F C700A1  469      STA      {RAM_area+self_mod+1-prog_rout}
0252 B68D     470      LDA      adr_lo
0254 C700A2  471      STA      {RAM_area+self_mod+2-prog_rout}
                                472
0257 B68B     473      LDA      eprom_data    ;get the data
                                474
0259      475      $IFNOT debug
0259 141C     476      BSET      latch,eprog    ;set lat bit
025B      477      $ENDIF
                                478
                                479      ;next instruction is modified by the program itself
                                480      self_mod:
025B C7FFFF   481      STA      $FFFFH      ;write data to EPROM
                                482
025E      483      $IFNOT debug
025E 101C     484      BSET      epgm,eprog    ;set pgm bit
0260      485      $ENDIF
                                486
0260 AE04     487      LDX      #pr_time     ;time counter in X
0262 4F      488      CLRA      ;reset A
                                489
                                490      prog1:
0263 9D      491      NOP      ;delay 256 * 4uS = 1 mS
0264 4A      492      DECA     ;8 cycles = 4 uS/ loop
0265 26FC    493      BNE      prog1
                                494
0267 5A      495      DECX     ;decrement X
0268 26F9    496      BNE      prog1      ;and loop till X = 0

```


PRITSE.ASM
Program-IT-Self
Program routines

Assembled with IASM 03/10/1994 10:13 PAGE 13

```

497
026A 151C      498          BCLR    latch,eprog    ;clear lat bit
026C 111C      499          BCLR    epgm,eprog     ;clear pgm bit
500
501 ;turn off VPP
026E 1100      502          BCLR    pgmr,porta
0270 macro     503          inport  pgmr,porta
0272 81        504          RTS
505 prog_end:
506
0273          507 prog_size EQU    {prog_end - prog_rout}
508
```

```
508
509 ;=====
510 ;=          Get data          =
511 ;=====
512 ; Function: get_data(,): data, carry
513 ; Description: get data from the serial line
514 ; Check that the input figures are OK
515 ; Input: none
516 ; Output: carry flag set if error
517 ; Uses: none
518
519 get_data:
0273 macro 520     message data_msg
0278 CD0201 521     JSR     recbyte
027B 2502   522     BCS     get1         ;quit if error
523
524 ;OK, good result
027D 98    525     CLC
027E 81    526     RTS         ;clear carry
                    ;OK, get back
527
528 ;Oh no, error
529 get1:
027F macro 530     message quest_msg
0284 99    531     SEC
0285 81    532     RTS         ;set carry
533
534
535 ;=====
536 ;=          Read block        =
537 ;=====
538 ; Function: read_blk(,)
539 ; Description: read_bl reads the data from EPROM
540 ; and sends it to the SCI port
541 ; Input: none
542 ; Output: none
543 ; Uses: bytcount, ROM_area, adr_lo, adr_hi,
544 ; colcount, bytcount
545
546 read_blk:
0286 3F86 547     CLR     bytcount     ;prepare to display 256 bytes
0288 A640 548     LDA     #{ROM_area & 0FFH}
028A B78D 549     STA     adr_lo      ;addr := #ROM_area
028C A603 550     LDA     #{ROM_area / 100H}
028E B78C 551     STA     adr_hi
552
553 readb2:
0290 macro 554     message buffer_msg     ;send buffer header
0295 A610 555     LDA     #16         ;prepare 16 columns
0297 B787 556     STA     colcount     ;16 bytes/line
557
558 readb3:
0299 CD023C 559     JSR     read         ;read the EPROM
029C CD01A1 560     JSR     to_ascii     ;write result in on terminal
029F 3C8D 561     INC     adr_lo      ;address:=address + 1
02A1 2602 562     BNE     readb4
02A3 3C8C 563     INC     adr_hi
564 readb4:
```

```
02A5 3A86      565      DEC      bytecount      ;check if end of message
02A7 2706      566      BEQ      readb1        ;done if yes
02A9 3A87      567      DEC      colcount      ;else check if end of colcount
02AB 27E3      568      BEQ      readb2        ;then output nl
02AD 20EA      569      BRA      readb3        ;else continue
570
571 readb1:
02AF 81        572      RTS                    ;Done, get back
573
574 ;=====
575 ;=          Find free      =
576 ;=====
577 ; Function: find_free(): carry
578 ; Description: find free EPROM byte for programming
579 ; Looks at an area that is 256 byte
580 ; large to find free byte
581 ; Input: none
582 ; Output: value in adr_hi, adr_lo.
583 ; Carry set if eprom is full
584 ; Uses: ROM_area
585
586 find_free:
02B0 AEFB      587      LDX      #0FFH          ;start at end of table
02B2 A600      588      LDA      #erased      ;look for non erased bytes
589
590 find2:
02B4 D10340    591      CMP      ROM_area,X    ;check to see
02B7 2608      592      BNE      find1        ;jump if the cell is
593 ;not empty
02B9 5A        594      DECX                    ;yes, decrement X
02BA A3FF      595      CPX      #$FF          ;
02BC 26F6      596      BNE      find2        ;jump back if X > 0
02BE 4F        597      CLRA                    ;
02BF 2004      598      BRA      find3        ;EPROM area is empty
599
600 find1:
02C1 9F        601      TXA                    ;the cell was not empty
02C2 4C        602      INCA                    ;add one to get first
603 ;empty cell
02C3 270B      604      BEQ      find4        ;exit if outside area
605
606 ;compute absolute address
607 find3:
02C5 AB40      608      ADD      #{ROM_area & 0FFH}
02C7 B78D      609      STA      adr_lo
02C9 4F        610      CLRA                    ;
02CA A903      611      ADC      #{ROM_area / 0FFH}
02CC B78C      612      STA      adr_hi
02CE 98        613      CLC                    ;clear carry flag
02CF 81        614      RTS
615
616 ;memory full, send error message
617 find4:
02D0 macro     618      message mem_full_msg
02D5 CD01E3    619      JSR      receive      ;wait for keypressed
02D8 99        620      SEC                    ;set carry flag
02D9 81        621      RTS
```

```

622
623 ;=====
624 ;=          Program EPROM          =
625 ;=====
626 ; Function: prog_eprom(,)
627 ; Description: relocate the program
628 ; from EPROM to RAM
629 ; Moves the program to the area
630 ; starting with the label "RAM_area"
631 ; The first part of the routine copies
632 ; the second part into RAM
633 ; and then calls it.
634 ; Input: address in adr_hi and adr_lo
635 ;      data in eprom_data
636 ; Output: none
637 ; Uses: eprom_data, RAM_area
638
639 prog_eprom:
02DA CD02B0 640     JSR     find_free      ;look for free space
02DD 2513   641     BCS     prog_eprom2   ;get out if it can't be found
642
02DF CD0273 643     JSR     get_data
02E2 250E   644     BCS     prog_eprom2   ;quit if error
645
02E4 B78B   646     STA     eprom_data    ;save the result of get_data
02E6 AE2A   647     LDX     #prog_size    ;no of bytes to relocate
648
649 prog_eprom1:
02E8 D60248 650     LDA     {prog_rout-1},X ;get data source
02EB E78D   651     STA     {RAM_area-1},X ;store in dest
02ED 5A     652     DECX
02EE 26F8   653     BNE     prog_eprom1   ;loop until routine is copied
654                                     ;into RAM
655
02F0 BD8E   656     JSR     RAM_area      ;call the routine in RAM
657
658
659 prog_eprom2:
02F2 81     660     RTS
661
662 ;=====
663 ;=          Initilazation routine    =
664 ;=====
665 ; Function: init(,)
666 ; Description: this is where the MCU starts when
667 ; power is applied.
668 ; Input: none
669 ; Output: none
670 ; Uses: porta
671 init:
02F3 macro 672     inport  pgmr,porta    ;turn of pgmr
02F5 1100   673     BCLR   pgmr,porta    ;turn off programming voltage
02F7 1600   674     BSET   led,porta    ;and LED pin
02F9 macro 675     outport led,porta
02FB 1200   676     BSET   scitr,porta  ;set SCI output pin
02FD macro 677     outport scitr,porta
678

```

```

02FF macro      679          message init_msg          ;welcome message
                680
                681      init1:
0304 09000C     682          BRCLR   key,porta,main    ;jump to main programme
0307 CC030A     683          JMP      programme        ;or continue with 'programme'
                684
                685      ;=====
                686      ;=          Programme routine          =
                687      ;=====
                688      ; Function: init(,)
                689      ; Input: none
                690      ; Output: none
                691      ; Uses: none
                692
                693      programme:
                694
                695      ;read routine
030A CD0286     696          JSR      read_blk          ;read and display the memory block
                697
                698      ;write routine
030D CD02DA     699          JSR      prog_eprom        ;programme the EPROM
0310 CC0304     700          JMP      init1          ;jump back again
                701
                702
                703      ;=====
                704      ;=          Main program              =
                705      ;=====
                706      ; Function: main(,)
                707      ; Description: blinks a LED as fast as programmed
                708      ; in the prog routine
                709      ; Input: none
                710      ; Output: none
                711      ; Uses: adr_lo, adr_hi, eprom_data, porta
                712
                713      main:
0313 CD02B0     714          JSR      find_free          ;get pointer to first
                715          ;free address
0316 B68D       716          LDA      adr_lo            ;point at last valid data
0318 4A         717          DECA
0319 B78D       718          STA      adr_lo
031B A1FF       719          CMP      #0FFH
031D 2602       720          BNE      main4
031F 3A8C       721          DEC      adr_hi
                722      main4:
0321 CD023C     723          JSR      read              ;read the EPROM content
0324 B78B       724          STA      eprom_data        ;save the result
                725
                726      ;Loop here till key pushed
                727      main1:
0326 BE8B       728          LDX      eprom_data        ;move result in X
0328 1700       729          BCLR   led,porta          ;turn on LED in port A
                730
                731      main2:
032A CD0104     732          JSR      bitwait          ;delay 1/2400 s
032D 5A         733          DECX
032E 26FA       734          BNE      main2
                735

```

PRITSE.ASM
Program-IT-Self
Main program

Assembled with IASM 03/10/1994 10:13 PAGE 18

```
0330 BE8B      736      LDX      eprom_data
0332 1600      737      BSET     led,porta      ;turn off LED
                                738 main3:
0334 CD0104    739      JSR      bitwait      ;delay 1/2400 s
0337 5A        740      DECX     ;any more loops to do ?
0338 26FA      741      BNE     main3       ;if yes, goto main3
                                742
033A 0800C7    743      BRSET   key,porta,init1 ;jump back if mode switch
033D CC0326    744      JMP     main1        ;jump back forever
                                745
                                746
                                747 ;empty ROM area
0340          748      ORG     $
                                749 ROM_area:
                                750
                                751 ;Mask Option Register
0900          752      ORG     mor_adr
0900 00        753      DB     mor
1FFE         754      ORG     reset_vector
1FFE 02F3     755      DW     init
                                756
                                757 END
                                758
                                759
```

Symbol Table

ADR_HI	008C
ADR_LO	008D
BELL	0007
BITCOUNT	0080
BITWAIT	0104
BITWAIT1	0106
BUFFER_MSG	014D
BYTECOUNT	0086
COLCOUNT	0087
COUNT	0088
CR	000D
DATA_MSG	0169
DDR	0004
DEL24	0086
DST_ADR	008A
END	2000
EPGM	0000
EPROG	001C
EPROM_DATA	008B
ERASED	0000
ESC	001B
FIND1	02C1
FIND2	02B4
FIND3	02C5
FIND4	02D0
FIND_FREE	02B0
GET1	027F
GET_DATA	0273
HALFBITWAIT	0100
HEX	0084

PRITSE.ASM
Program-IT-Self
Main program

Assembled with IASM 03/10/1994 10:13 PAGE 19


HEXSTR	0191
INIT	02F3
INIT1	0304
INIT_MSG	012D
KEY	0004
LATCH	0002
LED	0003
LF	000A
MAIN	0313
MAIN1	0326
MAIN2	032A
MAIN3	0334
MAIN4	0321
MEM_FULL_MSG	0172
MOR	0000
MOR_ADR	0900
MSG	012D
NL_MSG	0166
PGMR	0000
PORTA	0000
PROG1	0263
PROGRAMME	030A
PROG_END	0273
PROG_EPROM	02DA
PROG_EPROM1	02E8
PROG_EPROM2	02F2
PROG_ROUT	0249
PROG_SIZE	002A
PR_TIME	0004
QUEST_MSG	0163
RAM_AREA	008E
RAM_START	0080
READ	023C
READ1	023E
READ2	0248
READB1	02AF
READB2	0290
READB3	0299
READB4	02A5
READR1	0238
READ_BLK	0286
READ_END	023C
READ_ROUT	022E
READ_SIZE	000E
REC0	01E7
REC1	01EA
REC2	01F0
REC3	01FB
RECB1	0203
RECB2	0223
RECB3	0227
RECBYTE	0201
RECEIVE	01E3
REC_CHAR	0082
RESET_VECTOR	1FFE
ROM_AREA	0340
ROM_END	0900

PRITSE.ASM
Program-IT-Self
Main program

Assembled with IASM 03/10/1994 10:13 PAGE 20

ROM_START	0100
SAV_CHAR	0083
SCIREC	0002
SCITR	0001
SELF_MOD	025B
SRC_ADR	0089
STOPBIT	0002
STRPTR	0085
TO_ASCII	01A1
TO_HEX	01BF
TO_HEX1	01D5
TO_HEX2	01C5
TO_HEX3	01DF
TO_HEX5	01E1
TRA1	0124
TRA2	0128
TRA3	0116
TRANSMIT	010A
TR_CHAR	0081
XMITMSG	018 0
XMITMSG1	0190
XMITMSG2	0182

All products are sold on Motorola's Terms & Conditions of Supply. In ordering a product covered by this document the Customer agrees to be bound by those Terms & Conditions and nothing contained in this document constitutes or forms part of a contract (with the exception of the contents of this Notice). A copy of Motorola's Terms & Conditions of Supply is available on request.

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and  are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

The Customer should ensure that it has the most up to date version of the document by contacting its local Motorola office. This document supersedes any earlier documentation relating to the products referred to herein. The information contained in this document is current at the date of publication. It may subsequently be updated, revised or withdrawn.

Literature Distribution Centres:

EUROPE: Motorola Ltd., European Literature Centre, 88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England.

ASIA PACIFIC: Motorola Semiconductors (H.K.) Ltd., Silicon Harbour Center,

No. 2, Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong.

JAPAN: Nippon Motorola Ltd., 4-32-1, Nishi-Gotanda, Shinagawa-ku, Tokyo 141, Japan.

USA: Motorola Literature Distribution, P.O. Box 20912, Phoenix, Arizona 85036.

