# GMRT MONITOR AND CONTROL MODULE SOFTWARE

Mukund Gadgil,
Computer Division

Version 1, Revision 0

April 28, 1992

## INTRODUCTION

Monitor and Control Module (MCM) is the vital part of the COntrol and MOnitor System (COMOS) of GMRT. It is a single card data acquisition and control system. MCM cards are distributed at all the remote antennas and at the Central Electronics Building (CEB). All the Monitor and Control points terminate on different MCMs. Each MCM has 64 analog input channels and 16 opto-isolated digital output lines. The typical monitor points are LO levels,power supply voltages,switch status etc.Typically,each MCM scans the all/specified input analog channels,digitizes the signals and stores the data into the on-chip internal RAM. It can also output the control word at the digital output port. Each MCM is connected to an Antenna Base Computer (ABC) by means of a shared serial link. The serial link between the MCM and the ABC may be either a plastic fiber or a shielded twisted copper pair. Monitor program on MCM receives the data from the ABC via this serial link,decodes it,processes the data and sends the response back to the ABC. This is the basic function cycle of any MCM,the details of which will be made clear further in this report.

This report has been divided into four sections as follows:

1. <u>Overview of MCM Hardware</u> : In this section the hardware of the MCM card is described briefly. Also some details regarding the microcontroller itself are mentioned in order to make the software part understandable.

2. <u>Overview of Multiprocessor Serial Communication and Communication Protocols</u> : This section narrates the way two microcontrollers (ABC and MCM) communicate with each other and how they exchange the information among themselves.

3. <u>Overview of MCM Software</u> : This is a brief description of the MCM monitor program. It gives the summary of the MCM operation as far as software is concerned.

4. <u>Details of the Monitor Program</u> : Detailed description of the MCM Software is given here. It covers all the modules of the monitor program viz. main program, command code routines, interrupt service subroutines etc.

SECTION I

## OVERVIEW OF MCM HARDWARE

In order to understand the details of the software backbone of MCM, it is necessary to gain a brief knowledge about its hardware.Siemens VLSI Circuit - SAB 80C535 is the heart of the MCM system,which is an 8-bit Microcontroller from the 8031 family with on-chip peripherals viz. timer/counter, 6 X 8-bit ports, 8-bit A/D converter with programmable voltage references, serial interface and 256 bytes of Data Memory.Circuit diagram of the MCM card is as shown in Appendix A. In the current version of MCM hardware,the processor clock frequency is 3.6864 MHz,which results into the 3.26 $\mu$sec machine cycle time.

Input Section: The 64 input lines to MCM are divided into 4 groups of 16 lines by means of four 16-to-1 multiplexers (MUX). MUXes used are Analog Devices's monolithic chips ADG 506. The ADG506 has 16 input lines,out of which one can appear at the output depending on the status of the four control lines of the same. Outputs of the four muxes are in turn connected to four separate analog input lines of the on-chip ADC.

ADC Section:80C535 has an on-chip 8-bit A/D converter with 8 multiplexed analog inputs.It operates on the principle of successive approximation by means of capacitive discharge. With a 3.6864 MHz crystal used, the ADC conversion time is approximately 46 $\mu$sec.(14 processor machine cycles).

Output Section: 16 digital output lines of each MCM are TTL compatible. However, these outputs are opto-isolated. Desired 16-bit word can be output on these lines for the various control applications viz. electromechanical device control,power switching etc.

Internal Data Memory: It can be divided into three physically separate and distinct blocks: the lower 128 bytes of RAM,the upper RAM area(128 bytes) and the 128-byte special function register (SFR) area.While the latter SFR area and the upper RAM area share the same address locations,they must be accessed through different addressing modes as shown in Table I.

Table I: Internal Data Memory Map

| Address space | Locations | Addressing mode |
|---|---|---|
| Lower 128 bytes of RAM | 00H to 7FH | Direct/Indirect |
| Upper 128 bytes of RAM | 80H to FFH | Indirect |
| Special Function Regs. | 80H to FFH | Direct |

From programming point of view,the lower RAM area can be grouped into three address spaces:

1) A general purpose register area that occupies locations 0 through 1FH.
2) The next 16 bytes (locations 20H to 2FH) contain 128 addressable bits.
3) Locations 30H to 7FH may be used as a scratch pad area.
Appendix B describes the detailed usage of these 256 byte of internal data memory.
Serial Interface: The SAB 80C535 has two serial interfaces which are functionally nearly identical as far as asynchronous communication is concerned. While communicating with the ABC, out of these two channels, serial interface channel-0 is used in half duplex mode. There are four different modes in which the channel can be programmed, out of which Mode 3 (9 bit UART, variable baud rate) mode is programmed by the monitor program of MCM. In this mode 11 bits get transmitted: a start bit(0), 8 data bits(LSB first), a programmable 9th bit, and a stop bit(1). This mode of communication is purposefully selected in order to facilitate the multiprocessor communication in the situation where many MCMs are connected to a single ABC through a shared serial link. Multiprocessor communication will be explained later in the report.
Special Function Registers:(SFRs)These registers are located in the internal RAM. These are the registers through which the CPU interfaces with all its peripherals. All control and data transfers from and to peripherals use this register interface exclusively. The monitor program on the MCM has to properly program these registers for a well behaved interface.
General Purpose Registers: These are the lower 32 locations of internal RAM. They are grouped into four banks, with each bank consisting of 8 General Purpose Registers (GPRs). Only one of these banks can be enabled at a time. Two bits in the Program Status Word (PSW,is one of the SFRs) PSW.3 and PSW.4, select the active register bank. The 8 GPRs of the selected register bank may be accessed by register addressing in which the instruction opcode indicates which register is to be accessed. For indirect addressing R0 and R1 are used as pointers to address internal memory.

Table II: GPR Bank Selection

| PSW.4 | PSW.3 | Bank selected |
|---|---|---|
| 0 | 0 | Bank 0 |
| 0 | 1 | Bank 1 |
| 1 | 0 | Bank 2 |
| 1 | 1 | Bank 3 |

## SECTION II

## OVERVIEW OF MULTIPROCESSOR SERIAL COMMUNICATION AND COMMUNICATION PROTOCOLS

As it has been already mentioned, a single ABC card communicates with multiple MCM cards via a shared asynchronous serial link. The serial channel between ABC and MCM has to be thus time multiplexed. Each MCM card has got its own address which can be set through a DIP switch at the input port. Monitor program, at power on, reads the DIP switch settings (i.e. the address) and stores it into the data memory. Then the mode-3 of the serial interface-0 is selected. If the master ABC wants to transmit a block of data to one of the several salve MCMs, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that the programmable 9th bit is set to 1 in address byte and 0 in data byte. With mode-3 selected for MCM card, no interrupt is generated with the data bytes when bit SM20 = 1. (SM20 is one of the eight bits of the SFR S0CON.) An address byte, however, will interrupt all slave MCMs, so that each slave MCM can examine the received byte and see if it is being addressed. The initial part of the SIO handler does the same thing. It compares the address byte sent by the ABC to its own address stored in the internal data memory. If the address match is found, that particular slave MCM clears its SM20 bit and it can get further interrupts with the data bytes. After having received a complete message, the slave sets SM20 again. The slaves that were not addressed leave their SM20 set and go on about their business, ignoring the incoming bytes.

Communication Protocols: Communication protocols are nothing but the fixed formats of the packets that are transmitted by MCM and ABC. In order to extract the information out of these packets this definite packet structure is required. For example, while a packet is sent from ABC to MCM, MCM must know things like packet length, command code, checksum of the packet etc. Accordingly there are different packet filds viz. packet size, id-code, control word etc. which have a definite position and meaning in the packet. Exact protocols for the ABC to MCM and MCM to ABC packets are given in Appendix C.

## SECTION III

## OVERVIEW OF MCM SOFTWARE

In GMRT COMOS, ABC ultimately controls the operation of MCM in the sense that it can issue certain "commands" to it and get the response back. A detailed description of the commands can be found in the Appendix D. The basic functions of the MCM software are -

i) to accept the command from ABC,

ii) to execute the issued command and

iii) to transmit the response data to ABC.

The whole monitor program can be divided into 4 sections:

a] POST and Initializations,

b] Main program,

c] Mode routines and

d] Interrupt Service Routines.

These are described in brief below.

a]<u>POST and Initializations:</u> At power on , POST (Power On Self Test) and system initializations are done by the microcontroller. During POST, the CPU checks for the proper functioning of its peripherals. After successful completion of POST, initializations are done during which various SFRs, GPRs, and internal data memory locations are programmed to specific values.

b] <u>Main Program:</u> To start with, function of the main program is to jump into one of the 'mode routines'. After the packet (data bytes) is received completely by the MCM from ABC, program control is transferred to main program again. Main program then performs the following function:

1) Validates the checksum of the received packet.

2) Checks whether proper command code (each packet received from ABC contains a command code that corresponds to a specific command to be executed by MCM) and its arguments are received or not. If not, an error flag is raised.

3) Executes the command. Prepares the core part of the response packet accordingly.

4) Prepares the header portion of the frame,calculates the checksum, puts it at the tail of the frame and transmits the

response frame to ABC.

5) After transmission is over main programs reinitiates the timer, various data memory locations, SFRs, GPRs and program control is transferred back to the selected mode.

c] <u>Mode Routines:</u> In the first version of the MCM software, there are two modes which the monitor program can exhibit viz. Idle Mode and Scan Mode. In either of the modes a sub-part of the response frame is made ready, serial reception is enabled and then the program enters into the corresponding mode. In idle mode the program does nothing; it just waits for the command to be issued from the ABC. In scan mode, selected analog input channels are scanned, converted into digital form and stored into the RAM. This is a cyclic processes that repeats itself until a packet is received from ABC.

d] <u>Interrupt Service Routines(ISR):</u> There are two types of interrupts on the MCM card : SIO (Serial Input-Output) interrupt and timer interrupt. Accordingly, two ISRs exist.

1) SIO handler: For the 80C535 microcontroller an interrupt can be generated after each byte is received or transmitted. In the monitor program, both receive as well as transmit interrupts are enabled. There are no separate interrupts for receive/transmit interrupts. It means that the Interrupt Vector Address is the same for both the interrupts. Proper care has to be taken in the SIO handler so as to identify the interrupt source.

2) <u>Timer handler:</u> Monitor program enables the timer-0 interrupt at power on. Timer-0 is used to detect the receive time-outs. Timer-0 registers (TH0 and TL0) are loaded with a count value that produces an interrupt after every 5 ms. Timer-0 restarts after every byte is received from ABC and it is stopped after the last byte in the frame is received from ABC. If suppose MCM is expecting a byte from ABC and if it does not get a byte within 5 ms; the timer interrupt gets generated. In the timer handler ISR a flag is set which indicates the mode program to jump into the main program. Main program in turn reports the time-out error to ABC when a next non-timeout ABC-frame is received.

## SECTION IV

## DETAILS OF THE MCM MONITOR PROGRAM

What happens at power on ?

As is done by all the microcontrollers from the 8031 family, 80535 also starts executing the program from ROM location 0000H at power on reset. The first instruction has to be a long jump instruction because first fifty ROM locations are reserved as the interrupt vector addresses. In the first version of the monitor program this jump is decided to be at location 0030H. So the actual main program starts from this location. Initially internal RAM is checked by writing 1s and 0s into each location and by reading them back to see if byte has been written correctly or not. If ram test fails, LED on the card is turned on, port pin p4.0 is set to 1, and serial reception is disabled. After successful completion of ram test, stack pointer is loaded with the value 65H so that now stack area gets defined over the locations 65H to 7FH. After this stack allocation now the program is ready to call different subroutines. The first subroutine called is the timer check routine.

Timer Check Subroutine: In this subroutine both the timers, Timer-0 and Timer-1, are checked by putting some count value into the respective count registers and then timers are triggered on. After few 'nop's it is checked if counter values have decremented or not. If not, an error is reported locally by turning on the LED, setting the p4.1 bit and disabling the serial input.

After the successful completion of ram and timer checks, a subroutine called init_80535 is called which initializes the 80535's SFRs.

Init_80535: Following sequence of operations is carried out in this subroutine:

Select DPTR0 (0th bank of the dptr).

Disable both the timers.

Enable interrupts.

Set the interrupt priorities.( Highest priority to SIO interrupt, second highest priority to Timer-0 interrupt and least to ADC interrupt.)

Set the serial communication mode to mode-3.

Configure timer-1 registers to produce 9600 baud rate.

Configure timer-0 to produce 5 ms ticks.

Start timer-1.

return to the calling program.

All these settings are done through different SFRs viz. DPSEL,PCON,IEN0,IEN1,S0CON etc.

After these SFR initializations, GPRs and internal RAM area are also initialized through a subroutine 'init'.

Init: The upper and lower ram locations are cleared to all 0s. Thus automatically the entries in the 'mode table' are all set 0s which set the default mode to idle mode. All the flags are also cleared. LED is turned off, MCM address is obtained by reading the dip switch settings and is stored in one of the the ram locations. Register r0 of bank-0 points to the start of the ram area where packet from ABC is to be stored. Then register bank 1 is selected and r0 of this bank points to the 5th location of the ram area which is reserved for MCM to ABC packets. Thus the registers r0 from banks 0 and 1 are the pointers to specific memory locations which get passed to the required parts of the program.

Now the main program is ready to jump into one of the modes. Since at power on, idle mode is set, it jumps into idle mode.

Idle Mode: Following are the things done in this mode:

Prepare the header portion of the response (MCM to ABC) frame. This is done by putting the appropriate numbers at the memory locations pointed to by the memory pointers that in turn are passed by the main program. This header includes the length of logical packet, response code and the arguments. Serial reception is enabled along with the address interrupt. Then the program simply loops until the frame is received from ABC. While in the loop, program continuously checks if 'exit' flag is set or not. If set, it exits out of the loop and jumps into the main program.

Scan Mode: Header part of the packet is made ready similar to the idle mode. Program enters into receive mode, enables reception and address interrupt. Then the actual samplings of the channels start. After each sampling is over, it is checked if 'exit' flag is set or not. If set, all the remaining channels are digitized and then the program flow gets diverted back to the main program. If exit flag is not found to be set, sampling is done continuously.

Interrupt handling: While the monitor program is in one of the modes, the reception of the data from ABC is handled completely by the SIO isr (interrupt service routine). So the main program does not have to bother about the data reception. Only thing to be done by the main/mode programs is to check for different flags that get set by the isr.

Whenever an sio interrupt occurs, it is first seen whether it is a receive interrupt or a transmit interrupt. This can be found from the RI0 and TI0 flags; the flag that is set denotes the type of the interrupt. With a receive interrupt RI0 gets set and with a transmit interrupt TI0 gets set. For the transmit interrupt, nothing is done except clearing the TI0 flag and then the program control is transferred to the point where it was before the interrupt. With a receive interrupt, it is checked whether the byte received is the first byte of the packet. If it is, then that byte is compared with the address of the MCM card. If a match is found, then the corresponding flag is set, LED is turned on, and the data interrupt is enabled to receive further incoming bytes. Otherwise it simply returns to the mode program (without enabling the data interrupts.)

As soon as the address match is found, timer-0 is triggered to run. If the next data byte does not arrive within the stipulated time period of 5 ms, timer interrupt is generated. In the timer isr, timer-0 is halted, two flags viz. the timeout flag and exit flag are set, and then the program returns to the main/mode program. On the other hand if the next byte is received before the timeout occurs, timer-0 is stopped. Byte is stored, pointer is incremented, and the byte-counter (which is nothing but the no. of bytes received) is incremented. If this count is found to be greater than the maximum count, a flag is set and the pointer is decremented. Further it is checked whether full frame is received or not. If complete packet is received, again a flag is set. If not, isr sets the 'exit' flag whenever only five bytes of the packet are remaining.

<u>What happens after the program jumps into main routine from the mode routine</u> ?
Program jumps from the mode routine into main program whenever an 'exit' flag is found to be set. After the control is back to the main program, it waits for either frame (from ABC) to be over or timeout to occur. If timeout occurs, following steps are taken:

Serial input is disabled.
Address interrupt is enabled.
Receive mode is enabled.
All the GPRs and SFRs are reinitiated.
All the flags, except the timeout flag, are cleared.
Program control is transferred back to the previous mode.

In case the frame is found to be over, first thing that is done is to calculate the checksum. Calculated checksum is compared with the the checksum sent by the ABC. With a mismatch found a flag called 'csumerr' is set. Then the command code is extracted from the packet and it is compared with the existing command codes. If the command code is found to be an unknown one, error is reported by setting the 'icr' (inconsistent command received) flag. Program control is then transferred to that portion of the main program which sends the packet to ABC.
With a proper command code found, program jumps into the respective command-code routine. In each such routine again the validity of the arguments is checked. With the invalid arguments found, again the icr flag is set and program control is then transferred to that portion of the main program which sends the packet to ABC. With proper arguments found, that command is executed and it jumps into 'transmit' part of the program.

<u>Xmit</u> <u>Program</u>: Before the actual packet transmission starts, the final packet length is calculated, checksum of the whole packet is calculated and is put at the tail of the packet. Control word is also set according to the status of the different flags set previously. Then the complete packet is transmitted in polled fashion. After the transmission is over, program control is transferred back to the mode set recently.

# APPENDIX B

## MEMORY MANAGEMENT ON THE MCM

Lower RAM: Range : 00H to 7FH. Used as follows:

|  |  |
|---|---|
| 00H to 1FH | 32 registers.(4 banks of 8 GPRs) |
| 20H to 22H | 24 bits for different flags. |
| 23H | Used as a 'temp' location in the program. |
| 24H | Version byte |
| 25H | Counter for no. of bytes received from ABC. |
| 26H | Checksum calculated by MCM is stored here. |
| 27H | MCM address is stored at this maloc. |
| 65H to 7FH | Stack locations. |

**Thus in the lower RAM area, memory locations from 28H to 64H (60 locations) are free for use.**

Upper RAM: Packet received from ABC is stored in this area.

It extends from 80H to FFH.

|  |  |
|---|---|
| 80H to 91H | Frame received from ABC resides in this area. |
| 92H to 99H | Analog mask(8 bytes) stored here. |
| 9AH to 9BH | 16 bit of digital mask. |
| 9CH to 9FH | 32 bits of digital mask. |
| A0H to A7H | 8 bytes for mode table. |

Packet to be transmitted to ABC starts at location A8H. Maximum packet length (MCM to ABC) can be 5FH(i.e. 95 bytes). **So there is no free memory area in upper part.**

# APPENDIX C

## COMMUNICATION PROTOCOLS

Communication from ABC to MCM: Message from ABC to MCM consists of the following fields:

1) Packet size : 2 bytes, lower byte first.

2) Identifier Code : 1 byte.

3) Control Word : 2 bytes, lower byte first. Upper byte does not contain any information. Lower byte is the command code.

4) Argument Length : 2 bytes, lower byte first.

5) Actual arguments: These are the variable no. of bytes specified in the argument length.

6) Checksum :1 byte. It is the last byte of the frame.It is checksum of all the bytes in the frame.

Communication from MCM to ABC: Packet structure for MCM to ABC communication is as follows:

1) Packet size : 2 bytes, lower byte first.

2) Identifier Code : 1 byte.

3) Control Word : Lower byte first.Lower byte indicates the communication status

Bit 0 : Set when timeout.

Bit 1 : Set when checksum error.

Bit 2 : Set when an inconsistent command is received.

Bit 3 : Set when packet size is too large to fit into the allocated internal data memory buffer.

Upper byte indicates the no. of logical packets in the physical packet.

The physical packet header contains the above mentioned three fields. So the first logical packet starts at the sixth byte of the physical packet.

Logical packet has the following format:

1) Length of the logical packet : 2 bytes.

2) Response code : 1 byte. It is the command code to which MCM is responding.

3) Arguments : These are the variable no. of bytes and are nothing but the arguments sent by the ABC for that particular command.

At power on the self test results are available as a single argument byte with following bit mappings:

Bit 0 : Timer-0 test.

Bit 1 : Timer-1 rest.

Bit 2 : ADC test.

Bit(x)=1 => ERROR.

The last byte of the physical packet will be the checksum byte.

# APPENDIX D

## MCM Executable Commands

Basically, there are three types commands that MCM can execute: NULL COMMAND, SET COMMAND and READ COMMAND. The last two commands can have arguments and sub-arguments as explained below.

1] NULL COMMAND: Command code for this command is 00. Whenever this command is received, MCM does not do anything. It simply sends the response to ABC stating its current mode along with the self-test results and remains in that mode only. This command is useful as a handshake command when MCM is not being accessed frequently.

2] SET COMMAND: Command code = 01.This command has different arguments,sub-arguments and accordingly there are five different SET commands.

a) SET MODE IDLE : Argument -> MODE (00), Sub-argument -> IDLE (00).

This command sets the MCM in idle mode.
b) SET MODE SCAN : Argument -> MODE (00), Sub-argument -> SCAN (01).

This command sets the MCM in scan mode.

c) SET ANALOG MASK : For this command argument is the 'Analog Mask'(01) and sub-arguments are the actual bytes of the mask (8 bytes).

d) Set Digital Mask 16 Bit : Argument -> 'Digital Mask 16 bit' (02), followed by the 2 bytes of the digital mask.

e) Set Digital Mask 32 Bit : Argument -> 'Digital Mask 32 Bit' (03), followed by the 4 bytes of the digital mask.

3] READ COMMAND : Read command has the command code 02. READ also has five different commands:

a) Read analog mask : Command code -> 02.(02 => read)

Argument -> 00 (00 stands for analog mask).

b) Read digital mask 16 bit : Command Code -> 02.

Argument -> 01 (01 => digital mask 16 bit).

c) Read digital mask 32 bit : Argument -> 02.

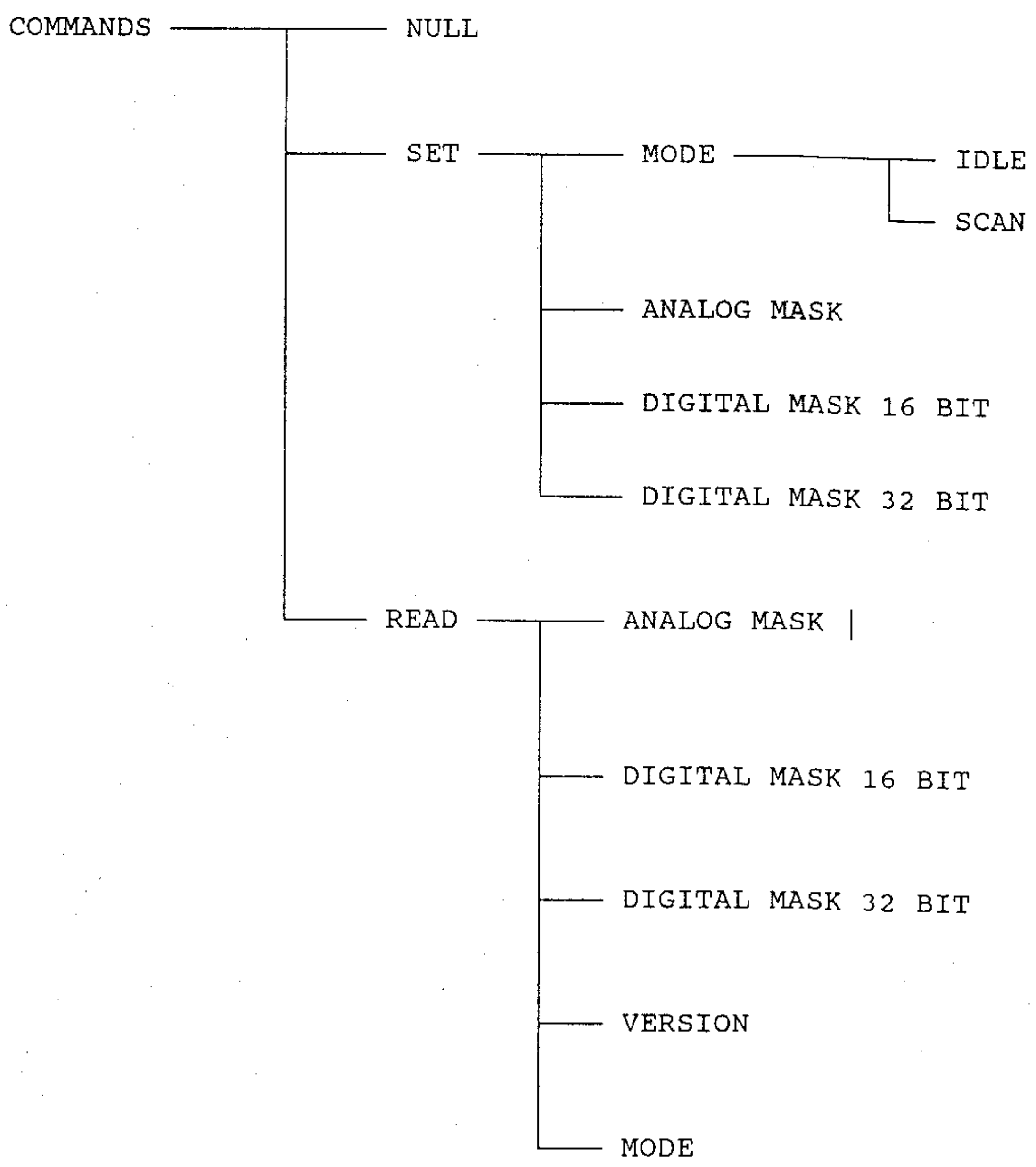
d) Read version : Argument -> 03. Reads the program version.

e) Read mode : Argument -> 04. Reads the current mode in which MCM is looping.

Number quoted against an argument is the actual hex code for that particular argument.

There are different buffers allocated for analog mask(8 bytes), digital mask 16-bit(2 bytes), digital mask 32-bit(4 bytes), mode(8 bytes) and version(2 bytes). Any SET command copies the respective arguments and sub-arguments into the respective buffers. For SET DIGITAL MASK command MCM outputs the 16/32 bits on the digital output port. Similarly any READ command MCM reads from the respective buffer and sends this data to ABC. Detailed packet formats for all these commands can be found in Appendix.

Tree diagram of MCM Commands :



# <u>Library of the subroutines/functions used in MCM Software Version 1</u>

1] **TMR_CHK : Timer test subroutine.**

```
Input  : None.
Output : A byte in accumulator.
           (a) = 00 ==> Timer test passed.
               = 01 ==> Timer test failed.
```

2] **ADC_CHK : On-chip ADC test subroutine.**

```
Input   : None.
Output  : A byte in accumulator.
            (a) = 0 ==> ADC test passed.
                = 1 ==> ADC test failed.
```

3] **INIT_80535 : Microcontroller specific initializations.**

```
Input  : None.
Output : None.
Action  :  This  subroutine when executed does the
           following things:
           i)   Selects the 0th dptr.
           ii)  Enables  and sets the priorities
                of different interrupts.
           iii)  Configures timers 0 and 1. Timer
                 0  for  generating 5 ms ticks to
                 detect  ABC  time-outs  and timer
                 1 to generate the 9600 baud.
           iv)  Initializes S0CON SFR in Mode-3.
```

4] **INIT : Software initializations.**

```
Input  : None.
Output : None.
Action :
          i)  Clears all the internal RAM locations
              to  00.  (This automatically sets the
              power-on mode to be idle mode.)
          ii) Clears all the flags.
          iii)Reads the MCM address.
          iv) I nitializes the memory pointers.
```

5] **SET_CW : Puts control word in the MCM to ABC packet.**

```
Input   :  i)  Pointer to the 'control-word' malloc.
           ii) Software flags.
Output  :   Proper control word set into 'control-
            word' malloc.
```

6] **GET_CHKSUM : Calculate the checksum of the packet.**

```
Input  : i)  Pointer to the start of the packet.
         ii) Packet length including checksum byte.
Output : i)  Checksum  of the packet. (Appends at
             the end of packet.)
         ii) Packet length in r1(1)
```

7] **SEND_BYTE :** Sends the byte on the serial link.

```
Input  : Byte to be transmitted in accumulator.
Output : TI0 bit set.
```

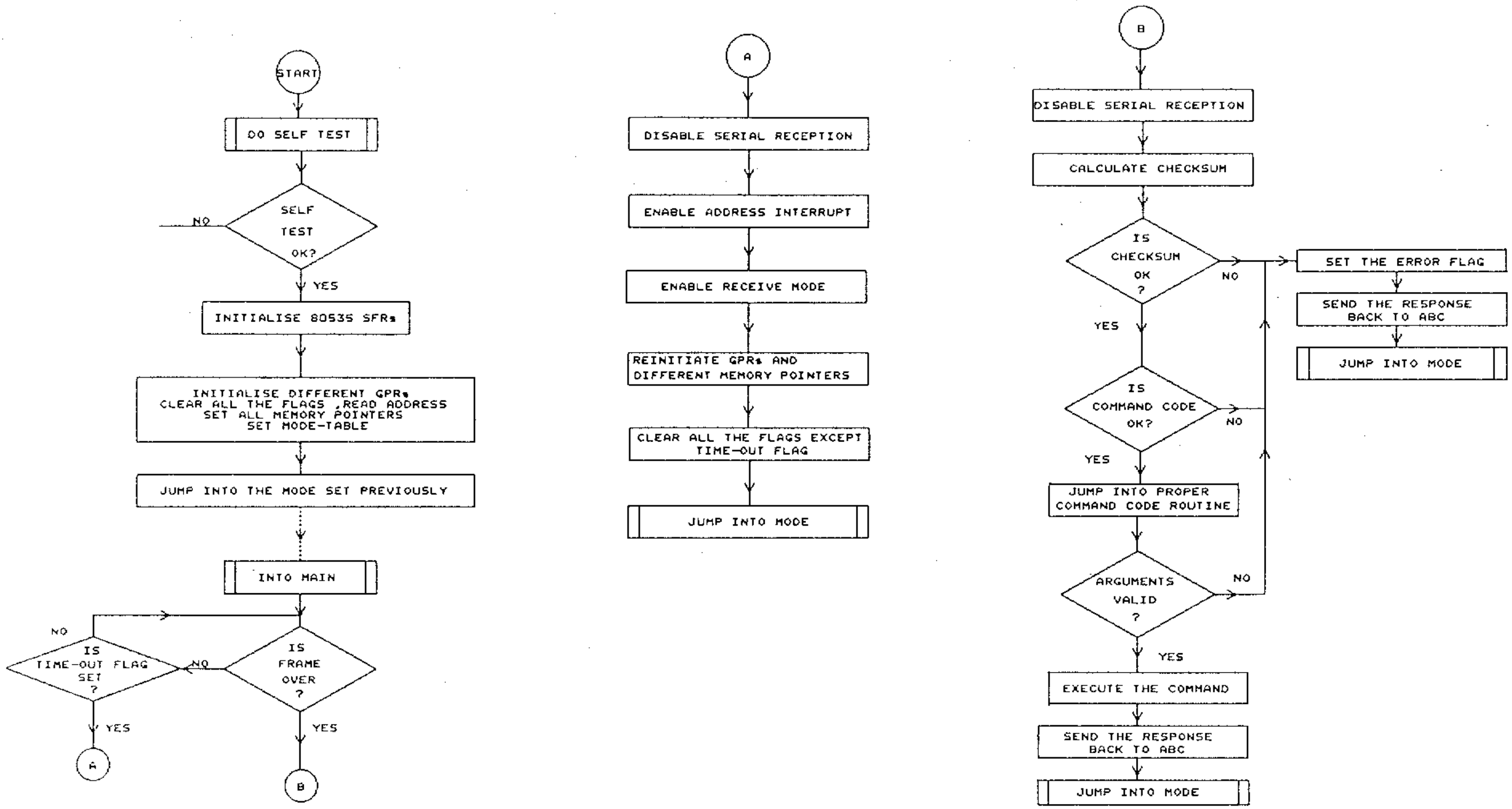
There are different segments of the program which are not used as the subroutines. Instead, they are the modules of the program. But these modules can be used as subroutines. Such modules are listed below:

8] **RAM TEST Function :** Tests the internal ram of the microcontroller. Alternate 0s and 1s are written into the ram locations and are read back and then checked with the bit pattern written in. If the two match through out the whole ram space, test is a success. Else it is declared to be
ram test failure.

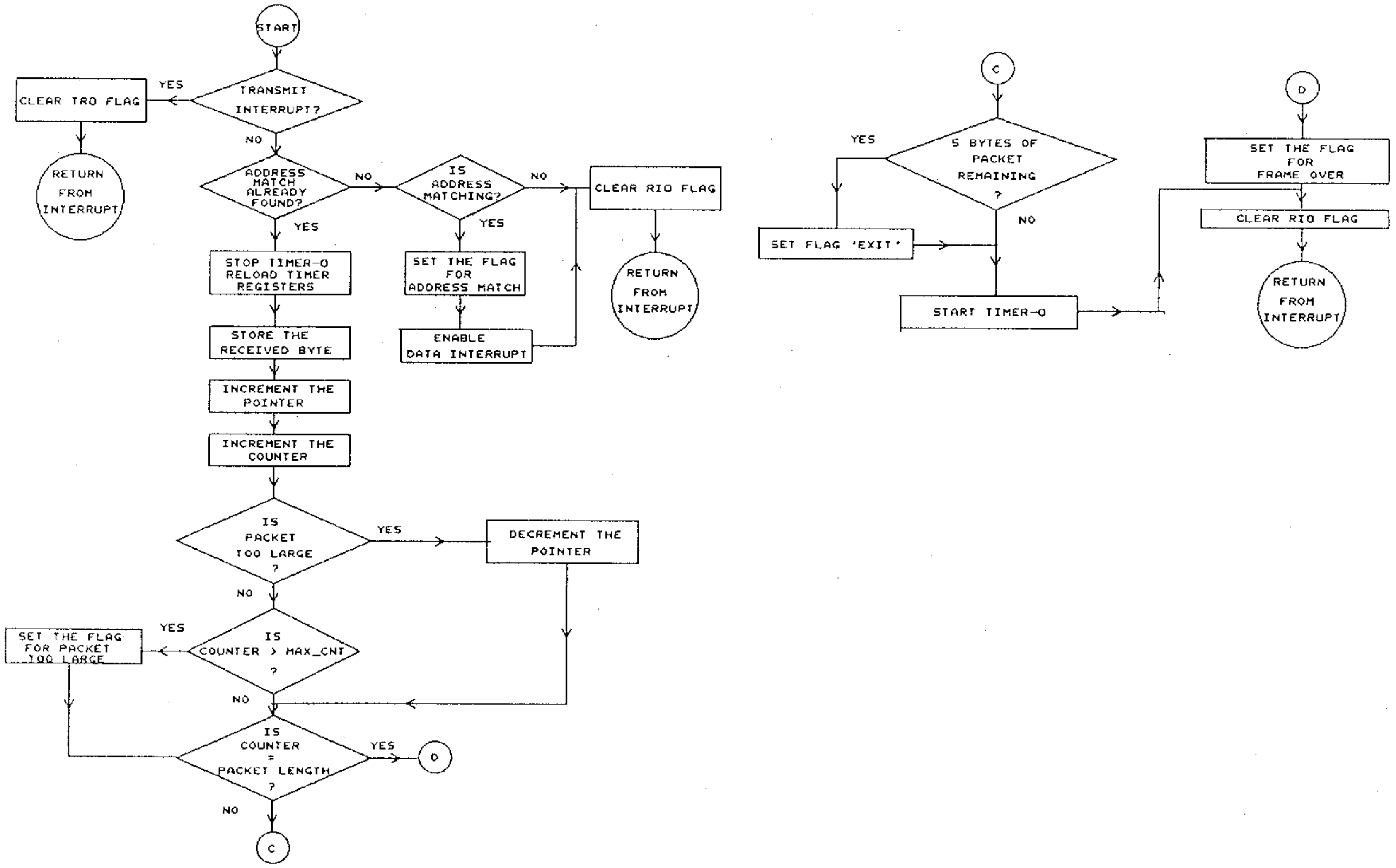
9] **GET_AD Function :** Configures the ADCON0 SFR (i.e. loads the proper byte into it) and DAPR SFR. Starts the conversion cycle of the ADC. SFR bytes can be passed as the i/p parameters and the output parameter is the contents of the ADDAT SFR after conversion is over.

10] **Xmit Function :** Input : Start of the packet, Packet length.
Output : None.

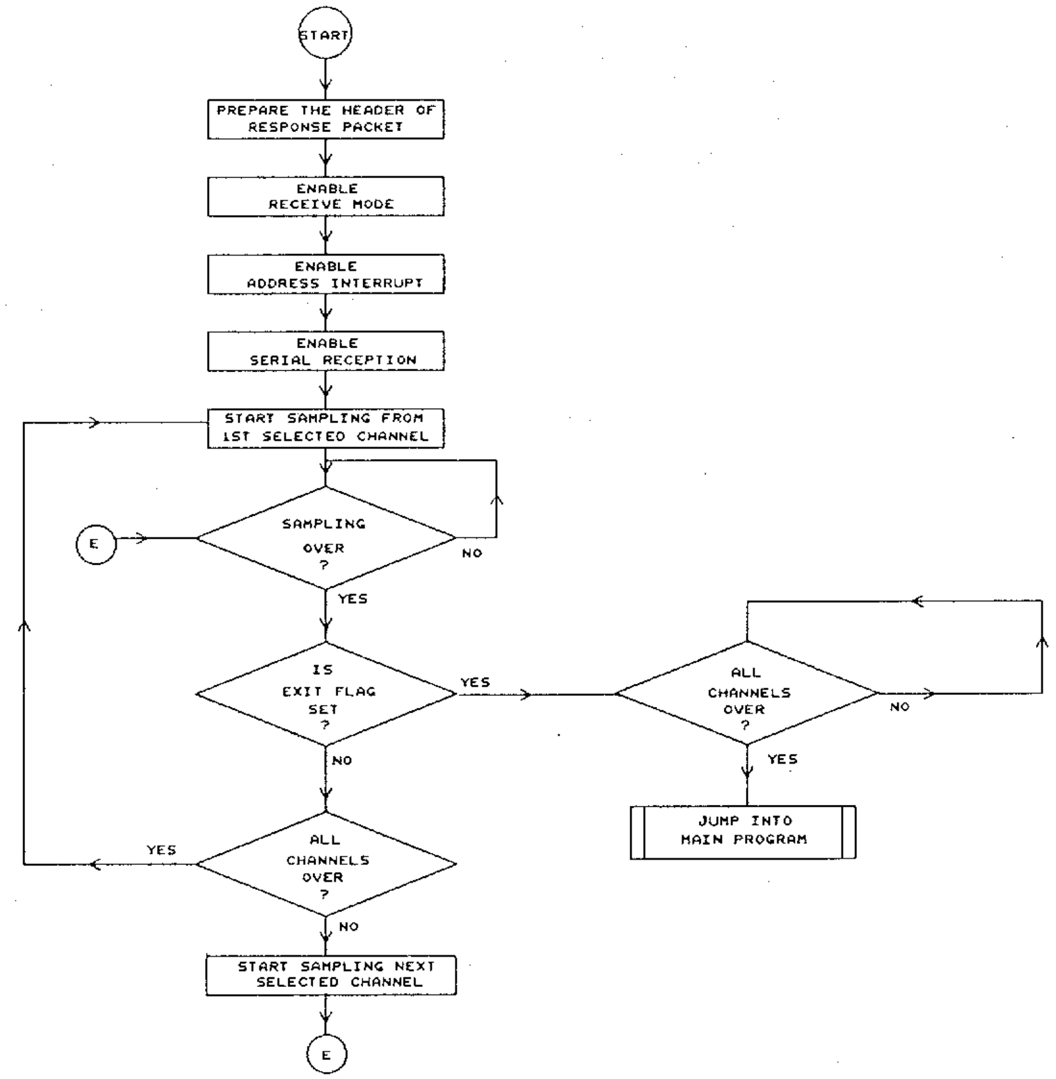
FLOWCHART FOR THE MAIN PROGRAM OF THE MCM MONITOR PROGRAM



FLOWCHART OF SIO ISR FOR THE MCM MONITOR PROGRAM



FLOWCHART FOR THE SCAN MODE OF MCM MONITOR PROGRAM



```
;               LISTING OF THE MAIN PROGRAM OF THE MCM SOFTWARE
;               -----------------------------------------------
NAME MAIN_PROG

        ;               PROGRAM FOR MONITOR AND CONTROL MODULE
        ;               --------------------------------------
        ;               program by MUKUND GADGIL
;-------------------------------------------------------------------------
;Initializations done in the file : INIT_MAB.ASM (for 3.6864 MHz crystal)
;-------------------------------------------------------------------------
        ;                   * 8/9-bit Format *
        ;           format_flag=0 => 8-bit format
        ;           format_flag=1 => 9-bit format

        table   SEGMENT         CODE
%SET(format_flag,1)      ;  /* 9-BIT FORMAT    */
        ;THIS IS THE MAIN MODULE
        ;-----------------------
        ;Symbols definitions

        pstart_in       EQU     80h             ;Starting common maloc area
                                                ;for received data
        init_byte_cnt   EQU     00h
        cntr_init       EQU     00h             ;Pointer initialisation value.
        stack           EQU     65h             ;Starting loc. for stack
        max_pack_in     EQU     19              ;Max pak_len.for ABC2MCM frame
        flag_strt       EQU     20h
        direct_buff     EQU     26h

        NULL_COMMAND    EQU     0
        SET_COMMAND     EQU     1
        READ_COMMAND    EQU     2
        REBOOT_COMMAND  EQU     3

        max_cmd_code    EQU     3
        max_set_code    EQU     4
        max_mode_code   EQU     2
        max_read_code   EQU     5

        mode_code       EQU     0
        scan_code       EQU     1


        restart         CODE    00H
        start0          CODE    30h
        sr_int          CODE    23H
        sr_int0         CODE    540h
        timer_0         CODE    0bh
        timer_0_jmp     CODE    600h
;       _iadc           CODE    43h
;       adc_service     CODE    700h
        ;-------------------------------------------------------------------
        ;RAM Storage locations for different packet fields of the frame are :-
        ;-------------------------------------------------------------------
        psize_lb        DATA    pstart_in       ;packet size lower byte
        psize_hb        DATA    psize_lb+1      ;packet size upper byte
        id_code         DATA    psize_hb+1      ;ID CODE of the MCM
        cmd_code        DATA    id_code+1       ;control_word_lb(cmd_code)
        control_word_hb DATA    cmd_code+1
        arg_lenth_lb    DATA    control_word_hb+1;Argument length lower byte
        arg_lenth_hb    DATA    arg_lenth_lb+1   ;Argument length upper byte
        args            DATA    arg_lenth_hb+1  ;Actual arguments are stored
                                                ;from this maloc

        args1           DATA    args+1
        args2           DATA    args1+1
        canl_mask       DATA    args2+8  ;Analog mask stored here
        dmask_16        DATA    canl_mask+8  ;16 bit digital mask stored here
        dmask_32        DATA    dmask_16+2  ;32 bit digital mask stored here
        mode_table      DATA    dmask_32+4  ;Current Mode byte (1 byte)
        pstart_out      DATA    mode_table+8  ;Start of the Response Packet
        flag_area       DATA    flag_strt
        temp_loc        DATA    direct_buff-2
        temp_locl       DATA    direct_buff-3
        version         DATA    direct_buff-1
        byte_cntr       DATA    direct_buff        ;Counter for no.of bytes
                                                   ;recceived from ABC.
        chk_sum         DATA    direct_buff+1      ;Checksum calculated by MCM
                                                   ;is stored here.
        mcm_addr        DATA    direct_buff+2      ;MCM    address stored here.


        tof             BIT     00h     ;Flag bit for time_out(20.0)
        csum_err_flag   BIT     tof+1
        tmr0_status     BIT     tof+2   ;Flag set when Timer0 NOT ok
        tmr1_status     BIT     tof+3   ;Flag set when Timer1 NOT ok
        tmr2_status     BIT     tof+4   ;Flag set when Timer2 NOT ok
        ram_status      BIT     tof+5   ;Flag set when internal ram check fails
        adc_status      BIT     tof+6   ;Set when ADC faulty
        come_out        BIT     tof+7   ;Flag which indicates the function to
                                        ;jump into the main programe.
        ptoo_large      BIT     tof+8   ;Set when a too large packet is
                                        ;received from ABC
        icr             BIT     tof+9  ;Flag for inconsistent command.
        slftst          BIT     tof+10  ;Flag for selftest.
        tot             BIT     tof+11
        null            BIT     tof+12  ;Flag set when mode is set to idle.
        ;----------------------------------------
        ;Actual code sitting in the ROM follows --
        ;----------------------------------------

        RSEG    table
;       Command Lookup Table :-
;       ---------------------

        lkupt:          DW      cmd_0   ;NULL COMMAND
                        DW      cmd_1   ;SET COMMAND
                        DW      cmd_2   ;READ COMMAND
;                       DW      cmd_3   ;REBOOT COMMAND

        set_table:      DW      set_mode
                        DW      set_anl_mask
                        DW      set_dmask_2b
                        DW      set_dmask_4b
;       Mode Lookup Table :-
;       ------------------

        modt:           DW      idle_mode
                        DW      scan_mode

        read_table:     DW      rd_anlmask
                        DW      rd_dmask16
                        DW      rd_dmask32
                        DW      rd_version
                        DW      rd_mode

        EXTRN   CODE(init_80535)                   ;init_80535 is an external
                                                   ;procedure
$ INCLUDE (reg535.pdf)
$ INCLUDE (rx2.asm)                     ;Include interrupt handeling

$ INCLUDE (init.pdf)
```

```
        CSEG    AT      restart

                ljmp    start

        ;Main program starts:-
        ;--------------------

        CSEG    AT      start0


        start:
                clr     REN0                    ;Disable reception
                mov     r0,#1
ram_loop1:      mov     a,#0aah
                mov     @r0,a
                mov     a,@r0
                cjne    a,#0aah,ram_err
                inc     r0
                cjne    r0,#0ffh,ram_loop1
                mov     @r0,#0aah
                mov     a,@r0
                cjne    a,#0aah,ram_err
                mov     r0,#1
ram_loop2:      mov     a,#55h
                mov     @r0,a
                mov     a,@r0
                cjne    a,#55h,ram_err
                inc     r0
                cjne    r0,#0ffh,ram_loop2
                mov     @r0,#55h
                mov     a,@r0
                cjne    a,#55h,ram_err
                sjmp    ram_over

ram_err:        clr     p3.4
                mov     p4,#01
                setb    ri0
                jmp     $

ram_over:       mov     sp,#stack
;               call    tmr_chk
;               call    adc_chk
                call    init_80535              ;Initialize the 80535 SFRs.

main_loop:      call    init
;-------------------------------------------------------------
;After 'init' subroutine is executed REG. BANK 1 is in use.
;-------------------------------------------------------------
                ljmp    into_mode

into_main:      jb      psw.5,frame_over
                jb      tof,tot_err
                sjmp    into_main

tot_err:
                clr     REN0                    ;Disable reception.
%if(%format_flag EQ 1) then (
                setb    s0con.5                 ;Set SM2 bit to get interrupt
                                                ;with address byte only.
                setb    p3.3                    ;Enter into receive mode
                )
        else    (
                setb    p3.3                    ;Enter into receive mode
                )
fi
                clr     psw.3
                clr     psw.4
                mov     r0,#pstart_in
                clr     psw.1
                clr     psw.5

                setb    psw.3                           ;REG. BANK 1
                mov     r0,#pstart_out+5
                mov     byte_cntr,#0
;               mov     th0,#0fdh
;               mov     tl0,#0
                mov     th0,#timer0_hb
                mov     tl0,#timer0_lb
                mov     version,#1
;               mov     p5,#2
                mov     flag_area,#0
                mov     flag_area+1,#0
                setb    tot
                ljmp    into_mode

frame_over:
                clr     REN0                            ;Disable reception.
;               setb    psw.4
;               setb    psw.3
;               mov     r1,#80h
;lll:           mov     r0,#0ffh
;               djnz    r0,$
;               djnz    r1,lll
;               mov     th0,#0
;               mov     tl0,#0
;               setb    TR0                             ;Start timer-0
                clr     psw.3                           ;Reg. bank 0
                clr     psw.4
                clr     p3.4                            ;Turn on the LED.

;-----------------------------------------------------------------------
;   After the full frame is received, reg. & memory status are as follows:
;   r0(0) -> points to the malloc immediately following the 'chk_sum'field
;   (byte_cntr) = total no.of bytes in the frame received from ABC.
;-----------------------------------------------------------------------
                dec     r0                              ;Store the chk_sum received
                                                        ;from ABC into proper field
                mov     chk_sum,@r0                     ;i.e. ("chk_sum").
;-----------------------------------------------------------------------
;Following fragment calculates the actual check sum.
;Calculated check sum will be in the Accumulator--
;-----------------------------------------------------------------------

                clr     a                               ;Clear the acc.
                mov     r1,byte_cntr                    ;Load r1 with total no. of
                                                        ;bytes received from ABC.
                dec     r1                              ;Decrement r1 by 1 because the
                                                        ;last i.e.'csum' field in the
                                                        ;frame has to be ignored while
                                                        ;calculating the check sum.
;               dec     r1
                mov     r0,#pstart_in                   ;Point to start of frame
        csum:   add     a,@r0
                inc     r0
                djnz    r1,csum
                cpl     a
                add     a,#1h                           ;Now calculated csum is in acc.
                cjne    a,chk_sum,csum_err              ;Mismatch between the 2 csums.
                                                        ;Report the error.

                sjmp    no_csum_err
```

```
csum_err:
                setb    csum_err_flag
                setb    psw.3
;               clr     psw.4
                mov     temp_loc,r0
                mov     r0,#pstart_out+3
                call    set_cw
                mov     r0,temp_loc
                mov     r6,#0
                ljmp    xmit

no_csum_err:

                setb    psw.3                        ;REG. BANK 1 IN USE
;               clr     psw.4
;               mov     temp_loc,r0
;               mov     r0,#pstart_out+3
;               call    set_cw
;               mov     r0,temp_loc
                setb    psw.4                        ;REG. BANK 3 IN USE.
                mov     r0,#cmd_code
                mov     a,@r0           ;Command Code is in the accumulator.
                clr     c
                subb    a,#max_cmd_code
                jnc     unknown_cmd
                mov     a,@r0
                cjne    a,#1,not_set
                sjmp    cmd_1
unknown_cmd:
                setb    icr
                clr     psw.4           ;REG.BABK 1
                mov     temp_loc,r0
                mov     r0,#pstart_out+3
                call    set_cw
                mov     r0,temp_loc
                mov     r6,#0
                ljmp    xmit

not_set:        ljmp    set_not

cmd_1:          setb    psw.4           ;REG.BANK 3
                mov     r1,#args        ;SET Command found.First check if its
                mov     a,@r1           ;argument is proper.r1(3)
                clr     c
                subb    a,#max_set_code
                jnc     unknown_cmd     ;Unknown argument code found.

                mov     a,@r1       ;Proper argument found.Jump to respective
                rl      a           ;routine.
                push    acc
                mov     dptr,#set_table              ;Point to the base address of
                                                     ;the command table
                movc    a,@a+dptr
                mov     r1,a                         ;r1 (1)
                inc     dptr
                pop     acc
                movc    a,@a+dptr
                mov     r2,a
                mov     dph,r1
                mov     dpl,r2
                clr     a
                jmp     @a+dptr         ;Jump to the proper SET command routine

set_mode:       ;setb   psw.3           ;REG. BANK 1
                clr     psw.4
                mov     r1,#args1        ;SET MODE found.First check if its
                mov     a,@r1            ;argument is proper.r1(1)
                clr     c
                subb    a,#max_mode_code
                jnc     unknown_cmd      ;Unknown argument code found.

                mov     temp_loc,r0
                mov     r0,#pstart_out+3
                call    set_cw
;               mov     r0,temp_loc

;               mov     temp_loc,r0
                mov     r0,#mode_table
                mov     r1,#args1
                mov     a,@r1
                mov     @r0,a
                mov     r0,temp_loc
                mov     @r0,#5           ;Length of 2nd Logical Packet (l.b.)
                mov     r6,#5
                inc     r0
                mov     @r0,#0           ;Length of 2nd Logical Packet (h.b.)
                inc     r0
                mov     r1,#cmd_code
                mov     a,@r1
                mov     @r0,a            ;Command code (Code:1 -> set)is put
                inc     r0
                mov     r1,#args
                mov     a,@r1
                mov     @r0,a            ;1st argument ( 0 -> mode ) put in lp
                inc     r0
                inc     r1               ;args1
                mov     a,@r1
                mov     @r0,a            ;2nd argument ( 0/1 -> idle/scan )
                inc     r0
                ljmp    xmit

set_anl_mask:
                setb    psw.3
                clr     psw.4
                mov     temp_loc,r0
                mov     r0,#pstart_out+3
                call    set_cw
;               mov     r0,temp_loc

;               mov     temp_loc,r0
                mov     a,#0
                mov     r0,#canl_mask
                mov     r1,#args1
anl_cpy:        push    acc
                mov     a,@r1
                mov     @r0,a
                pop     acc
                inc     a
                inc     r0

                inc     r1
                cjne    a,#8,anl_cpy

;               mov     r0,r2
                mov     r0,temp_loc
                mov     @r0,#12 ;Length of 2nd Logical Packet (Lower Byte)
                mov     r6,#12
                inc     r0
                mov     @r0,#0   ;Length of 2nd Logical Packet (Upper Byte)
                inc     r0
                mov     r1,#cmd_code
                mov     a,@r1
                mov     @r0,a           ;Command code (Code:1 -> set)is put
                inc     r0
                mov     r1,#args
                mov     a,#0
arg_cpy:        push    acc
                mov     a,@r1
                mov     @r0,a          ;1st argument (1 -> anl_mask ) put in lp
                inc     r0
                inc     r1              ;args1
                pop     acc
                inc     a
                cjne    a,#9,arg_cpy
                inc     r0
                ljmp    xmit

set_dmask_2b:
                setb    psw.3
                clr     psw.4
                mov     temp_loc,r0
                mov     r0,#pstart_out+3
                call    set_cw
;               mov     r0,temp_loc

;               mov     r2,r0
;               mov     temp_loc,r0
                mov     r0,#dmask_16
                mov     r1,#args1
                mov     a,#0
d16_cpy:        push    acc
                mov     a,@r1
                mov     @r0,a
                pop     acc
                inc     a
                inc     r0
                inc     r1
                cjne    a,#2,d16_cpy

                mov     p4,dmask_16
                mov     p5,dmask_16+1

;               mov     r0,r2
                mov     r0,temp_loc
                mov     @r0,#6  ;Length of 2nd Logical Packet (Lower Byte)
                mov     r6,#6
                inc     r0
                mov     @r0,#0   ;Length of 2nd Logical Packet (Upper Byte)
                inc     r0
                mov     r1,#cmd_code
                mov     a,@r1
                mov     @r0,a           ;Command code (Code:1 -> set)is put
                inc     r0
                mov     r1,#args
                mov     a,#0
arg_cpy1:       push    acc
                mov     a,@r1
                mov     @r0,a          ;1st argument (1 -> dmask_16 ) put in lp
                inc     r0
                inc     r1              ;args1
                pop     acc
                inc     a
                cjne    a,#3,arg_cpy1
                inc     r0
                ljmp    xmit

set_dmask_4b:
```

```
                setb    psw.3
                clr     psw.4
                mov     temp_loc,r0
                mov     r0,#pstart_out+3
                call    set_cw
;               mov     r0,temp_loc

;               mov     r2,r0
;               mov     temp_loc,r0
                mov     r0,#dmask_32
                mov     r1,#args1
                mov     a,#0
d32_cpy:        push    acc
                mov     a,@r1
                mov     @r0,a
                pop     acc
                inc     a
                inc     r0
                inc     r1
                cjne    a,#4,d32_cpy

                mov     p4,dmask_32             ;Set the 32 bit word to the
                mov     p5,dmask_32+1           ;ports.
                mov     p4,dmask_32+2
                mov     p5,dmask_32+3


;               mov     r0,r2
                mov     r0,temp_loc
                mov     @r0,#8  ;Length of 2nd Logical Packet (Lower Byte)
                mov     r6,#8
                inc     r0
                mov     @r0,#0   ;Length of 2nd Logical Packet (Upper Byte)
                inc     r0
                mov     r1,#cmd_code
                mov     a,@r1
                mov     @r0,a           ;Command code (Code:1 -> set)is put
                inc     r0
                mov     r1,#args
                mov     a,#0
arg_cpy2:       push    acc
                mov     a,@r1
                mov     @r0,a         ;1st argument (1 -> dmask_16 ) put in lp
                inc     r0
                inc     r1              ;args1
                pop     acc
                inc     a
                cjne    a,#5,arg_cpy2
                inc     r0
                ljmp    xmit

set_not:
                mov     r1,#cmd_code
                mov     a,@r1
                rl      a
                push    acc
                mov     dptr,#lkupt             ;Point to the base address of
                                                ;the command table
                movc    a,@a+dptr
                mov     r1,a                    ;r0 (0)
                inc     dptr
                pop     acc
                movc    a,@a+dptr
                mov     r2,a
                mov     dph,r1
                mov     dpl,r2
                clr     a
                jmp     @a+dptr                     ;Jump to the proper routine

cmd_0:
;-----------------------------------------------------------
;This is the NULL COMMAND.Program just sets the control word,
;returns the packet length(0,in this case)
;-----------------------------------------------------------
                setb    psw.3
                clr     psw.4
                mov     temp_loc,r0
                mov     r0,#pstart_out+3
                setb    null
                call    set_cw
                mov     r0,temp_loc
                mov     r6,#0

                ljmp    xmit

cmd_2:
;-----------------------------------------------------------------------------
;Following fragment of the program executes the command-2 i.e. READ COMMAND
;
;-----------------------------------------------------------------------------
                                         ;Read Command found.Check first if
                mov     r1,#args         ;proper argument is received for this
                mov     a,@r1            ;command.
                clr     c
                subb    a,#max_read_code
                jnc     unknown          ;Invalid arg.found.Error!
                setb    psw.3
                clr     psw.4
                mov     temp_loc,r0
                mov     r0,#pstart_out+3
                call    set_cw
                mov     r0,temp_loc

                mov     r1,#args         ;Valid argument found,proceed.
                mov     a,@r1
                rl      a
                push    acc
                mov     dptr,#read_table        ;Point to the base address of
                                                ;the command table
                movc    a,@a+dptr
                mov     r1,a                    ;r0 (0)
                inc     dptr
                pop     acc
                movc    a,@a+dptr
                mov     r2,a
                mov     dph,r1
                mov     dpl,r2

                clr     a
                jmp     @a+dptr                 ;Jump to the proper routine

unknown:        ljmp    unknown_cmd

rd_anlmask:
;               setb    psw.3
;               clr     psw.4
                mov     @r0,#12     ;Length of 2nd Logical Packet (Lower Byte)
                mov     r6,#12
                inc     r0
                mov     @r0,#0      ;Length of 2nd Logical Packet (Upper Byte)
                inc     r0
                mov     r1,#cmd_code
                mov     a,@r1
                mov     @r0,a
                inc     r0
                mov     r1,#args
                mov     a,@r1
                mov     @r0,a
                inc     r0

                mov     a,#0
                mov     r1,#canl_mask
put_msk:        push    acc
                mov     a,@r1
                mov     @r0,a
                inc     r0
                inc     r1
                pop     acc
                inc     a
                cjne    a,#8,put_msk
                ljmp    xmit

rd_dmask16:
;               setb    psw.3
;               clr     psw.4
                mov     @r0,#6      ;Length of 2nd Logical Packet (Lower Byte)
                mov     r6,#6
                inc     r0
                mov     @r0,#0      ;Length of 2nd Logical Packet (Upper Byte)
                inc     r0
                mov     r1,#cmd_code
                mov     a,@r1
                mov     @r0,a
                inc     r0
                mov     r1,#args
                mov     a,@r1
                mov     @r0,a
                inc     r0

                mov     a,#0
                mov     r1,#dmask_16
put_d16:        push    acc
                mov     a,@r1
                mov     @r0,a
                inc     r0
                inc     r1
                pop     acc
                inc     a
                cjne    a,#2,put_d16
                ljmp    xmit

rd_dmask32:
;               setb    psw.3
;               clr     psw.4
                mov     @r0,#8      ;Length of 2nd Logical Packet (Lower Byte)
                mov     r6,#8
                inc     r0
                mov     @r0,#0      ;Length of 2nd Logical Packet (Upper Byte)
                inc     r0
                mov     r1,#cmd_code
                mov     a,@r1
                mov     @r0,a
                inc     r0
                mov     r1,#args
                mov     a,@r1
                mov     @r0,a
                inc     r0
                mov     a,#0
                mov     r1,#dmask_32
put_d32:        push    acc
                mov     a,@r1
                mov     @r0,a
                inc     r0
                inc     r1
                pop     acc
                inc     a
                cjne    a,#4,put_d32
                ljmp    xmit
rd_version:
;               setb    psw.3
;               clr     psw.4
                mov     @r0,#5    ;Length of 2nd Logical Packet (Lower Byte)
                mov     r6,#5
                inc     r0
                mov     @r0,#0     ;Length of 2nd Logical Packet (Upper Byte)
                inc     r0
                mov     r1,#cmd_code
                mov     a,@r1
                mov     @r0,a
                inc     r0
                mov     r1,#args
                mov     a,@r1
                mov     @r0,a
                inc     r0
                mov     @r0,version
                inc     r0
                ljmp    xmit

rd_mode:
;               setb    psw.3
;               clr     psw.4
                mov     @r0,#5    ;Length of 2nd Logical Packet (Lower Byte)
                mov     r6,#5
                inc     r0
                mov     @r0,#0     ;Length of 2nd Logical Packet (Upper Byte)
                inc     r0
                mov     r1,#cmd_code
                mov     a,@r1
                mov     @r0,a
                inc     r0
                mov     r1,#args
                mov     a,@r1
                mov     @r0,a
                inc     r0
                mov     r1,#mode_table
                mov     a,@r1
                mov     @r0,a
                inc     r0
                ljmp    xmit


idle_mode:
;-------------------------------------------------------------
;Prepare the header portion of the response frame :-
;REGESTER BANK 1 IS IN USE.
;-------------------------------------------------------------
                mov     @r0,#4          ;llp(lower byte)
                inc     r0              ;Pointer to llp(upper byte)
                mov     r5,#4
                mov     @r0,#0          ;llp(upper byte)
                inc     r0              ;Pointer to Response Code
                mov     @r0,#reboot_command     ;Response code for
                                                ;Self Test/Reboot
                inc     r0              ;Pointer to arguments of the response
                                        ;code
                mov     @r0,#0
                mov     a,#1
                push    acc
                jnb     tmr0_status,t0ok

                orl     a,@r0
                mov     @r0,a
        t0ok:   pop     acc
                rl      a
                push    acc
                jnb     tmr2_status,t2ok
                orl     a,@r0
                mov     @r0,a
        t2ok:   pop     acc
                rl      a
                jnb     adc_status,adc_ok
                orl     a,@r0
                mov     @r0,a

        adc_ok: mov     a,@r0
                cjne    a,#0,stst_failed
                sjmp    stst_ok

stst_failed:    clr     p3.4
                mov     p5,#0aah
                setb    ri0
                jmp     $

stst_ok:        inc     r0        ;r0(1) points to the Start of the out_buff that
                                  ;will be filled by main prog. if needed
                mov     byte_cntr,#cntr_init
                clr     ri0             ;Enable the RI Interrupt
%if(%format_flag EQ 1) then (
                setb    p3.3                    ;Enter into receive mode
                setb    s0con.5                 ;Set SM2 bit to get interrupt
                                                ;with address byte
                )
        else   (
                setb    p3.3                    ;Enter into receive mode
                )
fi
                setb    psw.3
                setb    REN0                    ;Enable reception
                jnb     come_out,$      ;Wait till frame received
                ljmp    into_main


scan_mode:
;----------------------------------------------------------------------------
;REGESTER BANK USED : 1

;This command scans those channels for which the corresponding bit in the
;Analog Mask is set to 1. Others are skipped.
;Following are the registers used-
;r2:Counter for no. of mask-bytes covered.
;r3:Global counter for the no. of channel-bits covered
;r4:Counter for the actual no. of channels scanned.
;r7:Counter for no.of bits checked within the mask-byte
;r0:Points to the storage locations for ADC values for the different channels
;r1:Points to the analog mask location
;-----------------------------------------------------------------------------
                setb    psw.3
                clr     psw.4
                mov     r1,#canl_mask
                mov     r5,#0               ;It contains the no.of bytes to be
                                            ;returned by the mode to the main prog.
                mov     r2,#8
                mov     r3,#8
                mov     a,@r1
find:           rrc     a
                jc      scan
scan_ret:       djnz    r3,find
                mov     r3,#8
                inc     r1
                mov     a,@r1
                djnz    r2,find
                sjmp    lpl_got

scan:           inc     r5
                sjmp    scan_ret

lpl_got:
;               mov     p5,r5
                mov     a,r5
;               add     a,#3
;               add     a,#8
;               add     a,#11
                add     a,#13
                mov     r5,a
                mov     @r0,a               ;Logical Packet length (l.b)
                inc     r0
                mov     @r0,#0              ;Logical Packet length (h.b)
                inc     r0

                mov     @r0,#set_command ;Command code for SET
                inc     r0
                mov     @r0,#mode_code  ;Code for MODE
                inc     r0
                mov     @r0,#scan_code  ;Code for SCAN
                inc     r0
                mov     r1,#canl_mask
                mov     r2,#8
send_msk:       mov     a,@r1
                mov     @r0,a
                inc     r0
                inc     r1
                djnz    r2,send_msk

%if(%format_flag EQ 1) then (
                setb    p3.3                        ;Enter into receive mode
                setb    s0con.5                     ;Set SM2 bit to get interrupt
                                                    ;with address byte
                )
        else    (
                setb    p3.3                        ;Enter into receive mode
                )
fi
                mov     temp_loc1,p1
                orl     temp_loc1,#0fh  ;Make lower nibble of temp_loc1 HIGH

                clr     ADEX                ;Internal start of conversion
                clr     ADM                 ;Stop after 1 conversion.
                setb    REN0                ;Enable reception
                mov     temp_loc,r0         ;This is the maloc from where the
                                            ;ADC o/p data will be stored.
continue:       mov     r1,#canl_mask
                mov     r2,#08h             ;Counter for no. of mask-bytes covered.
                mov     r3,#00h             ;Global Counter for no.of channel-bits
```

```
                                        ;covered.
                  mov     r4,#00h       ;Counter for actual no.of channels
                                        ;scanned.
                  mov     r7,#08h       ;Counter for no.of bits checked within
                                        ;a mask-byte.
n_arg:            mov     a,@r1
          ;       clr     c
wait:     ;       rrc     a
          ;       jc      get_ad
                  rr      a
;                 mov     p4,r3
                  jb      acc.7,get_ad

return:           inc     r3            ;Increment the global counter
                  jb      come_out,chk_exit
not_over:         djnz    r7,wait
                  mov     r7,#8h        ;Reload the 'bit-counter' value.
                  inc     r1
                  djnz    r2,n_arg
                  mov     r0,temp_loc
                  sjmp    continue
chk_exit:
                  cjne    r3,#64,not_over ;Wait untill all channels over.
;                 cjne    r3,#63,not_over ;Wait untill all channels over.

                  ljmp    into_main

;---------------------------------------------------------
;Obtaining the ADC o/p for a channel is done below -
;---------------------------------------------------------
get_ad:
          ;       cpl     p5.7
                  inc     r4      ;Counter for actual no.of channels scanned
                  push    acc
                  mov     a,r3                  ;Bit counter value in acc.
                  mov     b,#16
                  div     ab                    ;After the division acc has
                                                ;the mux. no. & b=channel no.
                  orl     a,#0f8h
                  mov     p5,a
                  orl     adcon0,#07h           ;Set MUX0-2 bits to 1.
                  anl     adcon0,a              ;Proper mux no. has been set

                  mov     p4,adcon0

                  mov     a,b
                  orl     a,#0f0h               ;Make upper nibble HIGH
                  anl     a,temp_loc1
                  mov     p1,a
                  mov     dapr,#00h
                  jb      BSY,$
                  mov     a,ADDAT
                  cpl     a
                  mov     @r0,a
                  inc     r0
                  pop     acc
                  ljmp    return

xmit:
                  setb    psw.3                 ;REG. BANK 1
                  clr     psw.4
                  mov     r0,#pstart_out        ;Update the packet size value.
                  mov     a,r5
                  add     a,r6
                  add     a,#6
                  mov     @r0,a
```

```
                mov     r5,a    ;ppl l.b.(including csum byte) in r5(1)
                inc     r0
                mov     @r0,#0  ;ppl h.b.
                inc     r0
                mov     r1,#id_code
                mov     a,@r1
                mov     @r0,a
                call    get_chksum
;               clr     p3.3            ;Enter into Transmit mode
%if(%format_flag EQ 1) then (
                clr     p3.3
                setb    TB80            ;Bit (s0con.3) set for addr.
                )
        else    (
                clr     p3.3            ;Enter into transmit mode
                )
fi
;               clr     TR0             ;Stop timer-0.
                mov     a,mcm_addr      ;Address byte sent.
                call    send_byte
;               call    delay
                clr     TB80            ;Send Data Bytes now
;               mov     a,th0
;               call    send_byte
;               mov     a,tl0
;               call    send_byte
                mov     r0,#pstart_out
                mov     a,r2
                mov     r1,a            ;ppl in r1(1)
sending:        mov     a,@r0
                call    send_byte
;               call    delay
                inc     r0
                djnz    r1,sending
;               mov     a,th0
;               call    send_byte
;               mov     a,tl0
;               call    send_byte

                clr     psw.3
                clr     psw.4
                mov     r0,#pstart_in
                clr     psw.1
                clr     psw.5
                setb    p3.4                    ;LED Off!
                mov     flag_area,#0            ;Clear all the flags
                mov     flag_area+1,#0

                setb    psw.3                   ;REG. BANK 1
                mov     r0,#pstart_out+5
                mov     byte_cntr,#0
;               mov     th0,#0fdh
;               mov     tl0,#0
                mov     th0,#timer0_hb
                mov     tl0,#timer0_lb
                mov     version,#1
                clr     ri0
;               setb    REN0                    ;Enable recption
                clr     ti0
                ljmp    into_mode

get_chksum:
;-----------------------------------------------------
;Puts the checksum byte as the last byte of the packet.
;Also returns the packet length in reg. r1(1)
;-----------------------------------------------------
```

```
                mov     r0,#pstart_out
                dec     r5      ;This is the packet size without checksum byte
                mov     a,r5
                mov     r1,a    ;(ppl-1) in r1(1)
                inc     a
                mov     r2,a    ;Actual ppl in r2(1)
                clr     a
calc:           add     a,@r0
                inc     r0
                djnz    r1,calc
                cpl     a
                inc     a       ;Checksum in acc
                mov     r1,a    ;Checksum stored in r1
                mov     r0,#pstart_out
                mov     a,r5
        ;       inc     a       ;Packet size including checksum byte
                add     a,r0
                mov     r0,a
                mov     a,r1
                mov     @r0,a           ;Put the checksum byte at the end of
                                        ;the packet
                ret


send_byte:
                clr     ti0
                mov     sbuf,a          ;Get the byte in acc
                jnb     ti0,$
                ret
init:
                mov     r0,#pstart_in
                mov     r1,#128
        do:     mov     @r0,#00h
                inc     r0
                djnz    r1,do
                mov     r0,#direct_buff
                mov     r1,#50
        do1:    mov     @r0,#0
                inc     r0
                djnz    r1,do1

                clr     psw.3                   ;Use reg.bank 0
                clr     psw.4
                clr     psw.1
                clr     psw.5
                setb    p3.4                    ;LED Off!
                mov     flag_area,#0            ;Clear all the flags
                mov     flag_area+1,#0

                mov     a,p1
                anl     a,#0f0h
                mov     r0,#4
rot:            rr      a
                djnz    r0,rot
                mov     mcm_addr,a              ;Get the MCM Address
;               mov     p4,mcm_addr
                mov     r0,#pstart_in
                setb    psw.3                   ;REG. BANK 1
                mov     r0,#pstart_out+5
                mov     version,#1
                ret

set_cw:
;               setb    psw.3                   ;REG. BANK 1
;               clr     psw.4
```

```
                mov     a,#1
                mov     @r0,#0
                push    acc
                jnb     tot,no_tof
                orl     a,@r0
                mov     @r0,a
no_tof:         pop     acc
                rl      a
                push    acc
                jnb     csum_err_flag,no_csum
                orl     a,@r0
                mov     @r0,a
no_csum:        pop     acc
                rl      a
                push    acc
                jnb     icr,no_icr
                orl     a,@r0
                mov     @r0,a
no_icr:         pop     acc
                rl      a
                jnb     ptoo_large,ok
                orl     a,@r0
                mov     @r0,a
ok:             inc     r0
                jb      null,one_lp
                mov     @r0,#2                      ;No.of lp.s in pp
                ret
one_lp:         mov     @r0,#1
                ret


delay:
                setb    psw.4
                mov     r0,#0ffh
                djnz    r0,$
                clr     psw.4
                ret

into_mode:
                mov     r1,#mode_table  ;r1(1)
                mov     a,@r1
                rl      a
                push    acc
                mov     dptr,#modt                  ;Point to the base address of
                                                    ;the mode table
                movc    a,@a+dptr
                mov     r1,a                        ;r1 (1)
                inc     dptr
                pop     acc
                movc    a,@a+dptr
                mov     r2,a
                mov     dph,r1
                mov     dpl,r2
                clr     a
                jmp     @a+dptr                     ;Jump to the proper routine

tmr_chk:
;-----------------------------------------------------------------------------
;This subroutine tests the Timer-0 and Timer-1
;Testing is done by putting certain values in timer registers and then by
;checking wheather these values decrement or not.
;-----------------------------------------------------------------------------
                clr     TR0                         ;Disable Timer-0
                clr     TR1                         ;Disable Timer-1
                mov     th1,#0ffh                   ;9600 baud
                mov     tl1,#0ffh
                mov     th0,#0ffh
                mov     tl0,#0ffh
                mov     tmod,#gate_dable+timer+tmr1_mod2+tmr0_mod1
                setb    TR0                         ;Start Timer-0
                mov     r0,#7fh
                djnz    r0,$
                clr     TR0
                mov     r0,tl0
                cjne    r0,#0ffh,tmr0_ok

                mov     p4,#02h                     ;Timer-0 mal-functioning
                clr     p3.4
                setb    ri0
                jmp     $

tmr0_ok:        setb    TR1                         ;Start Timer-1
                mov     r0,#7fh
                djnz    r0,$
                clr     TR1                         ;Stop   Timer-1
                mov     r0,th0
                cjne    r0,#0ffh,tmr1_ok

                mov     p4,#03h                     ;Timer-1 mal-functioning
                clr     p3.4
                setb    ri0
                jmp     $

tmr1_ok:        ret

END
```

```
;       LISTING OF THE INITIALIZATION PART OF THE MCM SOFTWARE VERSION 1
;       ----------------------------------------------------------------------
NAME    up_80535

        prog_80535      segment         CODE
        RSEG            prog_80535

        %SET(format_flag,1) ; 8 BIT FORMAT

        EXTRN   DATA(p4,p5,s0con,pcon,ien0,ien1,ien2,ip0,ip1)
        EXTRN   BIT(BD)

        ;This program does the power on initializations for 80535
        ;------------------------------------------------------------

        ;------------------------------------------------------------
        ; USED FOR GENERATING 9600 BAUD WITH 3.6864 MHz CRYSTAL
        ;------------------------------------------------------------

        PUBLIC  init_80535
        USING   0
        init_80535:

$ include(init.pdf)
        start1:
;               mov     dpsel,#dptr0
                                                ;Select dptr0 for ext. mem.
                clr     TR1                     ;addressing.
                clr     TR0                     ;Disable timer1
                orl     PCON,#80h               ;Disable timer0
                mov     IEN0,#int_enable+s_port ;Set SMOD bit(PCON.7)
                                                 _int+t0_int_able ;Enable serial
        ;       mov     IEN1,#1                 ;Enable ADC Interrupt
        ;       mov     IEN1,#0                 ;Enable ADC Interrupt
                mov     IEN2,#00h
                mov     IP0,#11h  ;Highest priority to SIO Int.& second highest
                mov     IP1,#12h  ;priority to Tmr0 Int. & lowest to ADC Int.
%if( %format_flag EQ 1 ) then (

;               mov     S0CON,#sc_mod3+ser_ip_enable+mul_proc
                mov     S0CON,#sc_mod3+no_ser_ip+mul_proc

                )
        else    (

;               mov     S0CON,#sc_mod1+ser_ip_enable+mul_proc
                mov     S0CON,#sc_mod1+no_ser_ip+mul_proc

                )
fi
        ;Configure Timer1 in mode2 (auto reload) for proper baud rate :-
        ;------------------------------------------------------------------
                clr     BD
                mov     th1,#0feh               ;Baud rate enable (ADCON0.7)
;               mov     th0,#0fdh               ;9600 baud
;               mov     tl0,#0h                 ;For 5 ms time_out
                mov     th0,#timer0_hb
                mov     th0,#timer0_lb
                mov     tmod,#gate_dable+timer+tmr1_mod2+tmr0_mod1

                setb    TR1
                ret                             ;Start timer1

        END
```

LISTING OF THE isr FOR MCM SOFTWARE VERSION 1

```
;
;
        CSEG    at      sr_int                  ;Serial-int. vector location
                ljmp    service
        CSEG    at      sr_int0
        service:
                jb      ri,no_ti
                clr     ti0
;
                reti                            ;return

        ;-------------------------------------------------------------
        ;Following subroutine will be executed if RI interrupt is received
        ;
        ;input:sbuf contains the received byte
        ;
        ;-------------------------------------------------------------

        no_ti:
                jb      psw.1,not_1
                push    acc
                mov     a,sbuf
                cjne    a,mcm_addr,_end
                clr     p3.4
                setb    psw.1           ;Set the flag for address match found
                                        ;with data bytes.
%if(%format_flag EQ 1) then (
                clr     s0con.5
                )
        else    (
;
                clr     s0con.5         ;Clear SM2 bit,now interrupt will occur
                )
fi
                pop     acc
                mov     th0,#0fdh
                mov     tl0,#0
                clr     ri0
                setb    TR0             ;Start TMR0 only when valid address
                                        ;byte is received.
                reti

        _end:   pop     acc
                clr     ri0
                reti

        not_1:
                clr     TR0
                clr     psw.3           ;Switch to bank-0
        ;       clr     psw.4
                mov     @r0,sbuf        ;Store the received byte
                inc     r0              ;Increment the pointer
                inc     byte_cntr       ;Increment the 'byte-counter'
                push    acc

                jb      ptoo_large,len_nok
                mov     a,byte_cntr
;
                mov     p4,byte_cntr
                cjne    a,#max_pack_in,len_ok
                setb    ptoo_large
len_nok:        dec     r0
len_ok:         mov     r1,#psize_lb
                mov     a,@r1
                clr     c
                subb    a,byte_cntr
                cjne    a,#0,fnot_over

                setb    psw.5           ;Frame found to be over,set the flag.
                pop     acc
                setb    psw.3
                clr     tof
                clr     ri0
                clr     TR0
;               reti

fnot_over:
                cjne    a,#7,return1
                setb    come_out
return1:
                pop     acc
                setb    psw.3
                clr     tof
                mov     th0,#0fdh
                mov     tl0,#0
                clr     ri0
                setb    TR0
                reti

        CSEG    AT      timer_0

                ljmp    serv_tmr0

        CSEG    AT      timer_0_jmp

        serv_tmr0:

                clr     TR0                     ;Stop Timer-0
                clr     p3.4
                mov     th0,#0fdh               ;Reload the timer values.
                mov     tl0,#0
                setb    tof
                setb    come_out
                reti
```