CP/M System Test Drive:

The goal of this exercise is to take a PROM based CP/M system which is available from several configurations and use that infrastructure to learn the tools and accomplish something useful. It proves that one can do a lot early in the build cycle and that a CP/M system works just fine on a small scale. No question there is more power in a PC, but then that is not the reason for acquiring a CP/M system.

In this case, I am using the Lynch PROM which he provides as a debug tool for those who buy their first few project cards. This is the SW003 code discussed in his descriptions of the N8VEM hardware and software. The PROM monitor facility is the first tool for exploration. The CP/M system configuration provides a read only A: disk with Xmodem (XM), Real Time Clock (RTC), and Hello World (GM) programs, an smaller empty read-write B: disk, and a CP/M utility disk F: with all the tools you need to do something useful. Executing GM will send you a message and then return to the startup Monitor (not CP/M).

The Monitor provides a means for looking into the PROM and RAM space to see what is available and where these facilities are located. There is also the ability to read and set registers which can be used to configure and send data to the peripheral chips in the system (Parallel output 8255A and 16C550 UART for example). There is a R/W scratch register in the UART at 6FH, so one could load a hex value (o6fc3) and then go back and read the register (i6f) to see if the register was correctly loaded. So given the above, how about changing the baud rate with the load register facility? (This will fail because the UART is needed to execute the writes, and the change to the Line Control Register (80H) takes the UART from move character mode to configure mode so terminal communication is lost).

I used the CP/M dump tool and monitor facility to scan through the PROM chip to see what was built into that particular PROM. I could just as easily have used a PROM programmer to read and display the contents of the chip when I made myself a backup copy (and for a later exercise I can erase that full megabyte AT27C080 memory and build an even larger system while I run the board on a smaller half megabyte memory chip where I have several copies in case I do something dumb). The scan revealed the sign-on messages for each of the utilities included on the F: disk. This inclusion set is visually confirmed by requesting a Dir F: or LS F:.

```
F>dir
F: PIP      COM : ASM      COM : STAT     COM : DDT      COM
F: DUMP     ASM : DUMP     COM : SUBMIT   COM : XSUB     COM
F: ED       COM : LOAD     COM : DEBLOCK  ASM : RTC      COM
F: VDE263   COM : VINST263 COM : BOB      TXT : BBCBASIC COM
```

The first exercise is to use the Edit (ED) tool to examine and modify the BOB.TXT file. Ed is moderately intuitive once you get the hang of it, but it will take a while before one might call it user friendly. In any case, BOB is a good place to start, and with some functional comfort attained, there are the two ASM files to tackle. Use PIP to make a copy of DUMP.ASM (F>pip B:DECODE.ASM=F:DUMP.ASM), and then use the assembler (ASM) to build the DECODE.ASM file (generating DECODE.PRN and DECODE.HEX) and the LOAD tool to convert it to a DECODE.COM file [B>F:LOAD DECODE.HEX]. The COM file should execute in memory and produce the same result as the original DUMP tool. All of these exercises should be done from the B: disk (which is writable) with source disk designations to identify where the tools can be found on the read only F: disk [i.e. B>F:ASM F:DECODE.ASM]. The PRN and HEX files will be written to the default B: disk which is read-write (but relatively small).

The CP/M manual set (user and programmer) for CP/M 2.2 in the archive provide the detail command structure for all of these development tools. Given the small size of the B: disk, it will be necessary to ERASE (ERA DECODE.PRN) some of these files as you go along so as not to exceed the capacity of the RAM disk drive. Implementation of a disk support board (with or without a physical rotating disk) is a good way to gain more file space that can deal with large PRN and BAK files which are very useful during the development and debug efforts.

Once the basic utility of the CP/M tools is understood, it is a good time to develop and test a new utility from almost scratch. Almost scratch means that building a new tool need not start at square zero, but use chunks of already existing code to build on. For this exercise, let’s recall that this SW003 PROM configuration supports 9600 baud and no UART FIFO buffer; but there is no reason why we can’t change that. Look through the PROM source code with your PC and find the routines for setting the initial configuration of the UART chip and extract that code and the set of UART Register addresses (68H-6FH) to a separate file we will call BAUD.ASM. Go to the Web and find a PDF copy of the UART chip datasheet and compare the configuration data to the stub of code in your BAUD.ASM file.

Next, go back to CP/M on the B: disk and edit the DUMP.ASM file to remove all of the DUMP specific code, but keep the basic equates, addresses, and generic code that might be useful in the future to run I/O, or startup and return to CP/M code. This piece of code can serve as a skeleton to start new utility files or other useful programs. The SCRATCH.ASM file might look something like Appendix A.

Think about the functionality one might want in a routine to set UART baud rates on the fly. While we could locate the UART init code in the PROM and patch that code in the programmer buffer, we can build a tool for changing the baud rate after the system starts up. We will of course have to change the baud rate in the terminal emulator device, but there is no reason to go back and cold start CP/M again which would return the baud rate to the initial 9600 baud. One could elect to provide several common baud rates and turn on the FIFO while working with the UART. A simple program to accomplish the above task and return to CP/M via the warm boot path might look like Appendix B.

Implementing this code could be accomplished in several ways. Most direct, one might type the code into the BAUD.ASM file using the Editor, and then build the assembly file into an executable using the standard CP/M tools in the SBC. An ASM file could be developed on a PC and downloaded via the Xmodem (XM) tool on disk (B:A:XM R BAUD.HEX). Alternatively, the code could be put together and tested on a PC with a CP/M emulator (SIMH). The functional code could then be downloaded to the SBC using the Xmodem facility to move the HEX or COM version of the code. Somewhere along the way, it is worthy to master both approaches.

There is a very nice Altair 8800 emulator that is part of the Peter Shorn SIMH emulator system available on the WEB (here). This is a superset emulator for the Altair 8800 as its facilities exceed those of the original hardware system. There are a couple of 8MB virtual disk drives in the emulator system; which allow working with several large files concurrently. You can also easily load in your favorite non-CP/M tools, like a Z80 or relocatable assembler.

The next step to consider might well be mastery of the BASIC interpreter provided on disk F:. Having accomplished a baud rate change with an assembly code, it might well make sense to take this same requirement and implement it via the BASIC package. On the WEB, there is a site for the BBC Basic, which includes a CP/M executable, and a Windows executable. On that site there is a tab for RT Russell which will lead you to the Z80 CP/M version of Basic and there is a manual available (the manual on the first page is a 16bit windows version manual).

APPENDIX A

```
;       FILE XXXX PROGRAM, CODE SKELETON AS A STARTING PLACE
;
BDOS    EQU     0005H   ;DOS ENTRY POINT
CONS    EQU     1       ;READ CONSOLE
TYPEF   EQU     2       ;TYPE FUNCTION
PRINTF  EQU     9       ;BUFFER PRINT ENTRY
BRKF    EQU     11      ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
OPENF   EQU     15      ;FILE OPEN
READF   EQU     20      ;READ FUNCTION
;
FCB     EQU     5CH     ;FILE CONTROL BLOCK ADDRESS
BUFF    EQU     80H     ;INPUT DISK BUFFER ADDRESS
;
UART0:  EQU     68H     ;DATA IN/OUT REGISTER
UART1:  EQU     69H     ;RX STATUS
UART2:  EQU     6AH     ;INTERRUPTS REG
UART3:  EQU     6BH     ;LINE CONTROL
UART4:  EQU     6CH     ;MODEM CONTROL
UART5:  EQU     6DH     ;LINE STATUS
UART6:  EQU     6EH     ;MODEM STATUS
UART7:  EQU     6FH     ;SCRATCH REGISTER

;       NON GRAPHIC CHARACTERS
CR      EQU     0DH     ;CARRIAGE RETURN
LF      EQU     0AH     ;LINE FEED
;
;       FILE CONTROL BLOCK DEFINITIONS
FCBDN   EQU     FCB+0   ;DISK NAME
FCBFN   EQU     FCB+1   ;FILE NAME
FCBFT   EQU     FCB+9   ;DISK FILE TYPE (3 CHARACTERS)
FCBRL   EQU     FCB+12  ;FILE'S CURRENT REEL NUMBER
FCBRC   EQU     FCB+15  ;FILE'S RECORD COUNT (0-128)
FCBCR   EQU     FCB+32  ;CURRENT (NEXT) RECORD NUMBER (0-127)
FCBLN   EQU     FCB+33  ;FCB LENGTH
;
        ORG     100H
;
;       SET UP STACK
        LXI     H,0
        DAD     SP      ;ENTRY STACK POINTER IN HL FROM THE CCP
        SHLD    OLDSP   ;SET SP TO LOCAL STACK AREA (FINIS RESTORES)
        LXI     SP,STKTOP
;*****************************************************
;CODE EXTRACTED FROM RECENT PROM INIT ROUTINES
```

```
SIGNON: LD      A,C3H   ;ANY CHAR TO WRITE TO REG
        OUT     UART7,A ;WRITE TO SCRATCH REGISTER
        IN      A,UART7 ;READ IT BACK - IS REG THERE?
        CP      C3H     ;IF THERE THEN UART EXISTS
        JP      NZ,NO_UART   ;NO UART AT THIS ADDRESS

        LD      A,01H   ;SET EXIT STATUS CODE
        JP      UART_OK ;GO GET THE JOB DONE

NO_UART:LD      A,01H   ;SET EXIT ERR CODE
        JP      RESTART ;BACK TO SYSTEM

UART_OK:LD      A,80H   ;SET DLAB FLAG
        OUT     UART3,A ;
        LD      A,0CH   ;SET BAUD TO 9600
        LD      A,06H   ;SET BAUD TO 19200
        OUT     UART0,A ;SET THE UART TO RATE
        LD      A,00H   ;
        OUT     UART1   ;
        LD      A,03H   ;
        OUT     UART3,A ;
        JMP     FINIS   ;TO RETURN
;*************************************************
;
;       FILE NOT THERE, GIVE ERROR MESSAGE AND RETURN
        LXI     D,OPNMSG
        CALL    MSG
        JMP     FINIS   ;TO RETURN
;
OPENOK: ;OPEN OPERATION OK, SET BUFFER INDEX TO END
        MVI     A,80H
        STA     IBP     ;SET BUFFER POINTER TO 80H
;       HL CONTAINS NEXT ADDRESS TO PRINT
        LXI     H,0     ;START WITH 0000
;
;
FINIS:
;       END OF DUMP, RETURN TO CCP
;       (NOTE THAT A JMP TO 0000H REBOOTS)
        CALL    CRLF
        LHLD    OLDSP
        SPHL
;       STACK POINTER CONTAINS CCP'S STACK LOCATION
        RET             ;TO THE CCP
;
;
;       SUBROUTINES
```

```
;
BREAK:  ;CHECK BREAK KEY (ACTUALLY ANY KEY WILL DO)
        PUSH H! PUSH D! PUSH B; ENVIRONMENT SAVED
        MVI     C,BRKF
        CALL    BDOS
        POP B! POP D! POP H; ENVIRONMENT RESTORED
        RET
;
PCHAR:  ;PRINT A CHARACTER
        PUSH H! PUSH D! PUSH B; SAVED
        MVI     C,TYPEF
        MOV     E,A
        CALL    BDOS
        POP B! POP D! POP H; RESTORED
        RET
;
CRLF:
        MVI     A,CR
        CALL    PCHAR
        MVI     A,LF
        CALL    PCHAR
        RET
;
;
PNIB:   ;PRINT NIBBLE IN REG A
        ANI     0FH     ;LOW 4 BITS
        CPI     10
        JNC     P10
;       LESS THAN OR EQUAL TO 9
        ADI     '0'
        JMP     PRN
;
;       GREATER OR EQUAL TO 10
P10:    ADI     'A' - 10
PRN:    CALL    PCHAR
        RET
;
PHEX:   ;PRINT HEX CHAR IN REG A
        PUSH    PSW
        RRC
        RRC
        RRC
        RRC
        CALL    PNIB    ;PRINT NIBBLE
        POP     PSW
        CALL    PNIB
        RET
```

```
;
MSG:    ;PRINT GENERIC MESSAGE
;       D,E ADDRESSES MESSAGE ENDING WITH "$"
        MVI     C,PRINTF        ;PRINT BUFFER FUNCTION
        CALL    BDOS
        RET
;
;
GNB:    ;GET NEXT BYTE
        LDA     IBP
        CPI     80H
        JNZ     G0
;       READ ANOTHER BUFFER
;
;
        CALL    DISKR
        ORA     A       ;ZERO VALUE IF READ OK
        JZ      G0      ;FOR ANOTHER BYTE
;       END OF DATA, RETURN WITH CARRY SET FOR EOF
        STC
        RET
;
G0:     ;READ THE BYTE AT BUFF+REG A
        MOV     E,A     ;LS BYTE OF BUFFER INDEX
        MVI     D,0     ;DOUBLE PRECISION INDEX TO DE
        INR     A       ;INDEX=INDEX+1
        STA     IBP     ;BACK TO MEMORY
;       POINTER IS INCREMENTED
;       SAVE THE CURRENT FILE ADDRESS
        LXI     H,BUFF
        DAD     D
;       ABSOLUTE CHARACTER ADDRESS IS IN HL
        MOV     A,M
;       BYTE IS IN THE ACCUMULATOR
        ORA     A       ;RESET CARRY BIT
        RET
;
SETUP:  ;SET UP FILE
;       OPEN THE FILE FOR INPUT
        XRA     A       ;ZERO TO ACCUM
        STA     FCBCR   ;CLEAR CURRENT RECORD
;
        LXI     D,FCB
        MVI     C,OPENF
        CALL    BDOS
;       255 IN ACCUM IF OPEN ERROR
        RET
```

```
;
DISKR:  ;READ DISK FILE RECORD
        PUSH H! PUSH D! PUSH B
        LXI     D,FCB
        MVI C,READF
        CALL BDOS
        POP B! POP D! POP H
        RET
;
;       FIXED MESSAGE AREA
SIGNON: DB      'FILE TOOL VERSION 2.0$'
OPNMSG: DB       CR,LF,'NO INPUT FILE ON DISK$'

;       VARIABLE AREA
IBP:    DS  2   ;INPUT BUFFER POINTER
OLDSP:  DS  2   ;ENTRY SP VALUE FROM CCP

;       STACK AREA
        DS  64; ;RESERVE 32 LEVEL STACK
STKTOP:
;
        END
```

APPENDIX B

```
;       BAUD RATE PROGRAM, CHANGES UART BAUD RATE ON THE FLY
;
BDOS    EQU     0005H   ;DOS ENTRY POINT
CONS    EQU     1       ;READ CONSOLE
TYPEF   EQU     2       ;TYPE FUNCTION
PRINTF  EQU     9       ;BUFFER PRINT ENTRY
BRKF    EQU     11      ;BREAK KEY FUNCN (TRUE IF CHAR RDY)
OPENF   EQU     15      ;FILE OPEN
READF   EQU     20      ;READ FUNCTION
URTDIV  EQU     06H     ;19200 BAUD SETPOINT = 6
;
FCB     EQU     5CH     ;FILE CONTROL BLOCK ADDRESS
BUFF    EQU     80H     ;INPUT DISK BUFFER ADDRESS
;
;       UART REGISTER ADDRESS STRUCTURE FOR 16C550
UART0:  EQU     68H     ;DATA IN/OUT REGISTER
UART1:  EQU     69H     ;RX STATUS
UART2:  EQU     6AH     ;INTERRUPTS REG
UART3:  EQU     6BH     ;LINE CONTROL
UART4:  EQU     6CH     ;MODEM CONTROL
UART5:  EQU     6DH     ;LINE STATUS
UART6:  EQU     6EH     ;MODEM STATUS
UART7:  EQU     6FH     ;SCRATCH REGISTER

;       UART REGISTERS BY FUNCTION
;       MOSTLY READ OR WRITE, SOME READ & WRITE
URHR    EQU     68H     ;DATA RECEIVE REG
UTHR    EQU     68H     ;TRANSMIT DATA REG
UIER    EQU     69H     ;INTERRUPT ENABLE REG
UFCR    EQU     6AH     ;FIFO CONTROL REG
UISR    EQU     6AH     ;INTERRUPT STATUS REG
ULCR    EQU     6BH     ;LINE CONTROL REG
UMCR    EQU     6CH     ;MODEM CONTROL REG
ULSR    EQU     6DH     ;LINE STATUS REG
UMSR    EQU     6EH     ;MODEM STATUS REG
USPR    EQU     6FH     ;SCRATCH REG
UDLL    EQU     68H     ;DIVISOR LOWER BYTE
UDLM    EQU     69H     ;DIVISOR UPPER BYTE

;       NON GRAPHIC CHARACTERS
CR      EQU     0DH     ;CARRIAGE RETURN
LF      EQU     0AH     ;LINE FEED
;
;       FILE CONTROL BLOCK DEFINITIONS
FCBDN   EQU     FCB+0   ;DISK NAME
FCBFN   EQU     FCB+1   ;FILE NAME
FCBFT   EQU     FCB+9   ;DISK FILE TYPE (3 CHARACTERS)
FCBRL   EQU     FCB+12  ;FILE'S CURRENT REEL NUMBER
FCBRC   EQU     FCB+15  ;FILE'S RECORD COUNT (0 TO 128)
FCBCR   EQU     FCB+32  ;CURRENT/NEXT RECORD NUMBER (0-127)
FCBLN   EQU     FCB+33  ;FCB LENGTH
;
        ORG     100H
;
```

```
        ;       SET UP STACK
START:  LXI     H,0H    ;CLEAR HL THEN ADD IN ENTRY SP FROM CCP
        DAD     SP      ;ENTRY STACK POINTER IN HL FROM THE CCP
        SHLD    OLDSP   ;SET SP TO LOCAL STACK (FINIS RESTORES)

STACK:  LXI     SP,STKTOP ;JUST ABOVE END OF BAUD RATE UTIL CODE

SMESG:  LXI     D,SIGNON;REMIND US WHICH UTILITY THIS IS
        CALL    MSGS    ;MESSAGE SEQUENCER CODE

        ; UART DATA RETRIEVE, GET BAUD RATE VIA READING DIV REGS
UTEST:  MVI     A,0C3H  ;LOAD SOME DATA INTO SCRATCH REGISTER
        OUT     USPR    ;  SAVE IT TO HARDWARE REGISTER
        STA     SCRCH1  ;  SAVE IT TO MEMORY
        XRA     A       ;  CLEAR ACCUM FOR READBACK CHECK
        IN      USPR    ;  ORIGINAL REGISTER VALUE RETURNED?
        STA     SCRCH2  ;  STORE VALUE WE THINK WILL BE THE SAME

UTGET:  MVI     A,80H   ;DLAB ON TO ALLOW READ OF DIVISOR REGS
        OUT     ULCR    ;  DLAB "ON" VIA LINE CONTROL REGISTER
        XRA     A       ;  CLEAR ACCUM BEFORE LOADING DATA

RBAUD:  IN      UDLL    ;  READ DIVISOR (LSB)
        STA     DIVRG2
        MOV     E,A
        IN      UDLM    ;  READ DIVISOR (MSB)
        STA     DIVRG1
        MOV     D,A

NODLAB: MVI     A,03H   ;DLAB OFF LOCKS DIV REG ACCESS
        OUT     ULCR    ;  WRITE CONTROL REGISTER ACCESS LOCK

INREGS: IN      ULCR    ;DATA BITS, STOP BIT, PARITY STATUS
        STA     LINER   ;  SAVE THE LINE CONTROL REGISTER VALUE
        IN      UFCR    ;  READ IN THE FIFO CONTROL REGISTER
        STA     FIFOR   ;  STORE THE CONTENTS OF FCR

        ; START OF UART SETUP, SET BAUD RATE VIA DIV REGS
INITU:  MVI     A,80h   ;  UNLOCK DLAB TO WRITE DIVISOR REGS
        OUT     ULCR    ;  TURN ON DLAB FOR DIV REG

SBAUD:  MVI     A,06h   ;  DIVISOR FOR DESIRED BAUD RATE
        OUT     UDLL    ;  SET DIVISOR (LSB)
        MVI     A,00h   ;
        OUT     UDLM    ;  SET DIVISOR (MSB)

        ; SET LCR TO DEFAULT: LOCK DLAB AND SET CONFIG
SCONF:  MVI     A,03h   ; DLAB OFF, 8 DATA, 1 STOP, NO PARITY
        OUT     ULCR    ; LOAD CONFIGURATION

        ; SET MCR TO DEFAULT: SET HW FLOW CONTROL VIA MCR
SFLOW:  MVI     A,03h   ; DTR + RTS BOTH FORCED TO LOW STATE
        OUT     UMCR    ; LOAD FLOW CONTROL HANDSHAKE TO MCR

        ; SET FCR TO DEFAULT: ENABLE FIFO CONTROL OF DATA
SFIFO:  MVI     A,07h   ; ENABLE & RESET Tx & Rx FIFOS
        OUT     UFCR    ; EXECUTE FIFO CONTROL BY FCR
```

```
        XRA     A       ;
        IN      UFCR    ; CONFIRM CORRECT WRITTEN CONFIG
        STA     FIFORG  ;
        XRA     A       ;
        IN      USPR    ;RETRIEVE SCRATCH REG DATA
        STA     SCRCH3  ;  SEE IF STILL THERE
;       JMP     FINIS   ;BAIL OUT ON DONE (FALL THROUGH HERE)
;
FINIS:  ;BAIL-OUT PATH WITHOUT USING REBOOT VIA WARM/COLD BOOT
;       RETURN TO CCP AVOIDS REBOOT RESET BAUD RATES TO ORIGINAL
;       (NOTE THAT A JMP TO 0000H REBOOTS ALL HARDWARE)
        CALL    CRLF    ;SPACE AFTER MESSAGES PRINTED HERE
        LHLD    OLDSP   ;ADDRESS OF CCP STACK
        SPHL            ;SP CONTAINS CCP'S STACK LOCATION
        RET             ;TO THE CCP
;
;
;       MORE OR LESS STANDARD AND USEFUL SUBROUTINES
;
BREAK:  ;CHECK BREAK KEY (ACTUALLY ANY KEY WILL DO)
        PUSH H! PUSH D! PUSH B; ENVIRONMENT SAVED
        MVI     C,BRKF
        CALL    BDOS
        POP B! POP D! POP H; ENVIRONMENT RESTORED
        RET
;
PCHAR:  ;PRINT A CHARACTER
        PUSH H! PUSH D! PUSH B; SAVED
        MVI     C,TYPEF
        MOV     E,A
        CALL    BDOS
        POP B! POP D! POP H; RESTORED
        RET
;
CRLF:   ;STANDARD SKIP LINE ROUTINE
        MVI     A,CR
        CALL    PCHAR
        MVI     A,LF
        CALL    PCHAR
        RET
;
PNIB:   ;PRINT NIBBLE IN REG A
        ANI     0FH     ;LOW 4 BITS
        CPI     10
        JNC     P10
;       LESS THAN OR EQUAL TO 9
        ADI     '0'
        JMP     PRN
;
;       GREATER OR EQUAL TO 10
P10:    ADI     'A' - 10
PRN:    CALL    PCHAR
        RET
;
PHEX:   ;PRINT HEX CHAR IN REG A
        PUSH    PSW
        RRC
```

```
        RRC
        RRC
        RRC
        CALL    PNIB    ;PRINT NIBBLE
        POP     PSW
        CALL    PNIB
        RET
;
MSGS:    ;PRINT ERROR/SIGNON MESSAGE
;       D,E ADDRESSES MESSAGE ENDING WITH "$"
        MVI     C,PRINTF        ;PRINT BUFFER FUNCTION
        CALL    BDOS
        RET
;
GNB:    ;GET NEXT BYTE
        LDA     IBP
        CPI     80H
        JNZ     G0
;
;       READ ANOTHER BUFFER
        CALL    DISKR
        ORA     A       ;ZERO VALUE IF READ OK
        JZ      G0      ;FOR ANOTHER BYTE
;       END OF DATA, RETURN WITH CARRY SET FOR EOF
        STC
        RET
;
G0:     ;READ THE BYTE AT BUFF+REG A
        MOV     E,A     ;LS BYTE OF BUFFER INDEX
        MVI     D,0     ;DOUBLE PRECISION INDEX TO DE
        INR     A       ;INDEX=INDEX+1
        STA     IBP     ;BACK TO MEMORY
;       POINTER IS INCREMENTED
;       SAVE THE CURRENT FILE ADDRESS
        LXI     H,BUFF
        DAD     D
;       ABSOLUTE CHARACTER ADDRESS IS IN HL
        MOV     A,M
;       BYTE IS IN THE ACCUMULATOR
        ORA     A       ;RESET CARRY BIT
        RET
;
SETUP:  ;SET UP FILE
;       OPEN THE FILE FOR INPUT
        XRA     A       ;ZERO TO ACCUM
        STA     FCBCR   ;CLEAR CURRENT RECORD
;
        LXI     D,FCB
        MVI     C,OPENF
        CALL    BDOS
;       255 IN ACCUM IF OPEN ERROR
        RET
;
DISKR:  ;READ DISK FILE RECORD
        PUSH H! PUSH D! PUSH B
        LXI     D,FCB
        MVI C,READF
```

```
        CALL BDOS
        POP B! POP D! POP H
        RET
;
;       FIXED MESSAGE AREA
SIGNON: DB      'BAUD RATE CHANGE VER 1.0$'

;       VARIABLE AREA
IBP:    DS  2   ;INPUT BUFFER POINTER
OLDSP:  DS  2   ;ENTRY SP VALUE FROM CCP
DIVBR:  DS  2   ;VALUE STORED IN DLL/DLM REGISTERS
;
FIFOR:  DS  1   ;VALUE STORED IN FIFO CONTROL REGISTER
SCRCH1: DS  1   ;VALUE WRITTEN TO SCRATCH REGISTER
SCRCH2: DS  1   ;VALUE WRITTEN TO SCRATCH REGISTER
SCRCH3: DS  1   ;VALUE WRITTEN TO SCRATCH REGISTER
LINER:  DS  1   ;VALUE STORED IN LINE CONTROL REGISTER
;
DIVRG2: DS  1   ;
DIVRG1: DS  1   ;
FIFORG: DS  1   ;

;       STACK AREA
        DS  64; ;RESERVE 32 LEVEL STACK
STKTOP:
;
        END
```