Multiphrom

Multiphrom is a serial rom emulator for use with the TMS5220 speech chip, designed to get the most possible out of the BBC micro speech system. This document summarises what I have learnt about the speech system, and describes how to use Multiphrom.

Phrom Numbers

The Speech System User Guide implies that the system can access up to 16 phroms, numbered 0–F, each holding 16Ki of data. In fact, both the TMS5220 and OS1.2 use a 20 bit address, accessing up to 64 phroms; the limitation is the TMS6100 serial rom, which ignores the top two bits. Multiphrom uses all 20 bits, emulating 64 phroms numbered -64 to -1 or &FFC0 to &FFFF. If you use a genuine TMS6100 chip it will appear four times, as &Cn, &Dn, &En, and &Fn.

Phrom Formats

There are three data formats used in TMS6100 roms for the Beeb, and rom dumps are found in both bit orders. I will refer to the order used in the Speech System User Guide (which puts the Acorn copyright message in the Kenneth Kendal rom into Ascii) as Normal order, and the opposite as Reversed order, and I will describe all data in Normal order. Note that most pointers are reversed, but most other data is not.

Acorn Format

This is the format described in the Speech System User Guide, and used in the Kenneth Kendal rom (CM62024). The rom starts with a 64 byte header which is completely ignored by the speech system (though used by the *ROM filing system), then continues exactly as Indexed Format. In fact, if you use word numbers less than 32, OS1.2 will interpret the header as part of the index and produce random noise.

Indexed Format

This is the format used in the US phrom (VM61002), and the speaking clock phrom (VM71003A). In practice, this is the format assumed by OS1.2 for speech data, since it is identical to Acorn format for word numbers greater than 31. To speak words in this format, use

```
SOUND ph, wn, 0, 0
```

where ph is the phrom number (&FFC0–&FFFF) and wn the word number.

Address	**Data**
0	00
1	word count reversed (not checked by OS1.2)
2	word 1 address low byte reversed
3	word 1 address high byte reversed
⋮	
2n	word n address low byte reversed
2n+1	word n address high byte reversed

The rest of the phrom is speech data exactly like Unformatted phroms.

Unformatted Phroms

This is the format used by the other three easily available phroms (VM61003, VM61004, VM61005) and is just raw data, starting at address zero, one word starting at the first byte after another ends. To speak words in this format, use

```
SOUND ph-&40, wadd, 0, 0
```

where ph-&40 is the phrom number minus 64 (&FF80–&FFBF) and wadd the address of the start of the word (relative to the start of the phrom).
The data is a sequence of frames which can be 4, 11, 19, or 50 bits long, packed into bytes from high bit to low bit, and byte aligned only at word boundaries. Each frame is a sequence of up to twelve parameters, each of 3,4,5 or 6 bits, which index standard tables. I believe each parameter is in Normal order, but I haven't checked this.

Loading Phroms into Multiphrom

You can load any phrom into any slot in multiphrom. Create a file containing the raw binary data at the root level of a micro-SD card, with the name Ph_XXN if the bytes are in normal order, or Ph_XXR if they are reversed, where XX is the slot you want to put it in (C0-FF). Insert the card into multiphrom and the led will light; when it goes out you can remove the card and use the speech system. Each slot takes about 5 seconds to load, so it is a good idea to delete or rename phrom files after loading.
If you have a genuine TMS6100 installed, it will take up four slots. Multiphrom should detect it and disable itself for those slots, but I haven't been able to test this properly, as I don't have a TMS6100. You can still load data into those slots, but not use it until you remove the TMS6100.

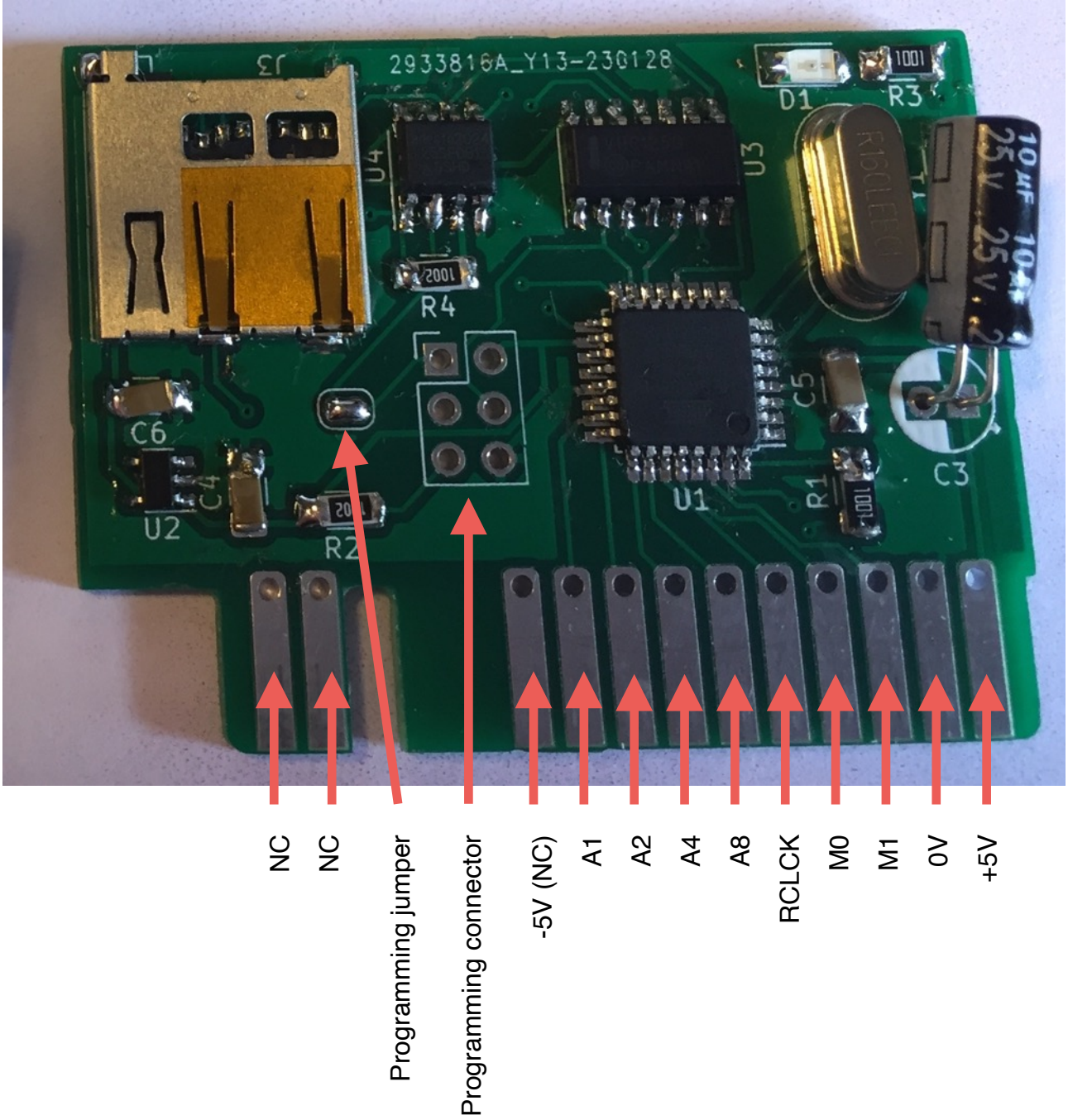
Using the *ROM Filing System

In OS1.2, the *ROM FS only searches slots &F0–&FF, so capacity is limited to 256KiB including overheads. It would be a simple patch (I believe only one byte) to allow all 64 slots for 1MiB storage; if any original cartridges exist they would then be repeated four times in the catalogue, which doesn't seem much of a downside. The format is as described in the New Advanced User Guide, all in normal order, with a header as described in the Speech System Users Guide with pointers in reversed order. I have written a BASIC program to generate appropriate phrom files, but files running over more than one slot should be considered beta, as there is no good documentation, and I haven't had time to test it thoroughly.

The Board, V2.0

The board contains an Atmega 168pb micro-controller, an AT25SF081 flash rom, a micro-SD card socket, and various components to connect them together. It is designed to plug into the cartridge socket to the left of the keyboard (the "ashtray") on a BBC micro. If you don't have a socket fitted in your Beeb, I recommend buying a 30 pin edge connector (Amp 5530843-2) and pulling out all the contacts on the left hand side, and the third, fourth and fifth from the top on the right, then put a key in the third position. This can be done with fine nosed pliers, and the socket will then solder to the keyboard PCB. WARNING: you may need fix PCB errors on either the main board or the keyboard; check the instructions in the BBC service manual for details. Failure to fix these errors may fry the board.

The board can also be plugged into PL14 (the second keyboard connector) on the Beeb either with a female header connector or a cable but WARNING: the pins are not quite in the same order. The -5V line is shifted from one end to the other, and connecting this wrong may fry the board. The simplest solution is to connect only the other nine lines, since -5V is not used by the board. If you decide to do this, do check the schematics carefully!



After testing, the flash rom should contain the UK speech phrom in slot FF, the US speech phrom in slot F0, the US speaking clock phrom in slot EF, and some BASIC programs (in *ROM format) in slot FE.

If you ever want to update the firmware, first desolder the programming jumper with desoldering braid, to protect the 3.3V chips from the 5V programming signals. The programming connector is a standard (Arduino style) ISP6 connector, and any Atmel-compatible programmer should work. After programming, remember to resolder the programming jumper.

Questions for Beta Testers

- Does it work reliably? Can you use it in all the expected ways?
- Can you find interesting new ways to use it? Are there straightforward improvements I could make to make it more useful?
- How easy is it to install in a Beeb? Which connection system works best?
- (If you have a genuine TMS6100) Does the automatic detection of genuine phroms work? You can test this with SOUND -17 for the UK phrom, or -32 for the US phrom: if the detection works, this should produce the installed phrom, just like SOUND -1 or -16. If it doesn't, I recommend removing the TMS6100 when using Multiphrom.

The Programs

I've included some simple programs which use the speech rom, in case you find them useful. These are installed in slot FE (using PHSAVE) so you can load them using *ROM. Feel free to use or adapt them in any way you like.

PHDUMP

PHDUMP generates a hex dump of the contents of any phrom slot. Just input the slot number in hex, and it will print out the contents, formatted for 80 column modes. For 40 column modes, change the 16 in line 200 to 8 (or however many bytes per line you prefer).

input slot number, and ensure it's meaningful:

```
 10 PRINT"Input slot number (C0-FF)";
 20 INPUT SN$
 30 SN=EVAL("&"+SN$) AND &3F
100 @%=4
```

set bottom twelve bits of speech address to zero:

```
110 *FX 159,0,64
120 *FX 159,0,64
130 *FX 159,0,64
```

set top six bits of address to slot, and next two to zero:

```
140 A%=159:X%=0:Y%=64+SN*4 MOD 16:R%=USR(&FFF4)
150 A%=159:X%=0:Y%=64+SN DIV 4:R%=USR(&FFF4)
```

loop through entire phrom slot:

```
160 FOR I%=0 TO &3FFF
```

tell TMS5220 to read byte:

```
170 *FX 159,0,16
```

read byte from phrom:

```
180 A%=158:R%=USR(&FFF4)
190 Y%=((R% DIV &10000)-1) AND &FF
```

print newline & address every 16 bytes, then read byte:

```
200 IF I% MOD 16 = 0 THEN PRINT:PRINT~I%," ",;
210 PRINT ~Y%,;
```

loop & end

```
220 NEXT
230 END
```

PHLIST

PHLIST searches through all installed phroms looking for those in Acorn format, recognised by the copyright message which starts at byte 2. This is useful to find which slots contain *ROM files.

loop through the 64 slots:

```
100 FOR PN%=0 TO 63
```

set the bottom 14 address bits to &0002 and the top six to the slot number:

```
110 *FX 159,0,66
120 *FX 159,0,64
```

```
130 *FX 159,0,64
140 A%=159:X%=0:Y%=64+PN%*4 MOD 16:CALL(&FFF4)
150 A%=159:X%=0:Y%=64+PN% DIV 4:CALL(&FFF4)
```

if the next three characters are not “(C)” go on to the next slot:

```
160 IF FNREADBYTE <> &28 OR FNREADBYTE <> &43 OR FNREADBYTE <> &29
THEN NEXT:END
```

read the copyright and title strings, and print them out:

```
170 C$="(C)"+FNREADSTRING
180 T$=FNREADSTRING
190 PRINT ~PN%+&C0;": ";T$;" “;C$
```

go on to next slot:

```
200 NEXT:END
```

function to read a byte at the current address:

```
210 DEF FNREADBYTE
220 *FX 159,0,16
230 A%=158:R%=USR(&FFF4)
240 = ((R% DIV &10000)-1) AND &FF
```

function to read a 0-terminated string at the current address:

```
250 DEF FNREADSTRING
260 R$=""
270 REPEAT:C=FNREADBYTE
280 IF C=0 THEN UNTIL TRUE: =R$
290 R$=R$+CHR$(C)
300 UNTIL FALSE
```

PHINDEX

PHINDEX searches through an unindexed phrom to find the start address of words. You need the address of the first word; it then scans through, assuming each word immediately follows the last. This technique allowed me to generate indexes of all the readily available TI roms.
Input the slot number (in hex) of the phrom you want to index, then the start address (in hex) of the first word (usually 0); it will then print the address and speak the word, before prompting “More?”. You can press return to continue the search, type a new address, or type N to stop.

input slot number & start address, and make meaningful:

```
100 INPUT"Slot number (C0-FF)",SN$
110 SN=EVAL("&"+SN$) AND &3F
120 INPUT"Start address (Hex)",SA$
130 SA=EVAL("&"+SA$) MOD &4000
```

keep track of how many bits are read:

```
135 BIT%=0
140 REPEAT
150 PRINT "Trying “;~SA
```

speak the next word (if it is one) & confirm to continue the search:

```
160 SOUND&FF80+SN,SA,0,0
170 INPUT "More",Q$:IF LEFT$(Q$,1)="N" THEN END
175 IF Q$<>"" THEN SA=EVAL("&"+Q$) MOD&4000:UNTIL FALSE
```

set the 20-bit speech address:

```
  180 A%=159:X%=0:Y%=&40+(SA AND &0F):CALL(&FFF4)
  190 A%=159:X%=0:Y%=&40+(SA DIV &10 AND &0F):CALL(&FFF4)
  200 A%=159:X%=0:Y%=&40+(SA DIV &100 AND &0F):CALL(&FFF4)
  210 A%=159:X%=0:Y%=&40+(SN*4+SA DIV &1000 AND &0F):CALL(&FFF4)
  220 A%=159:X%=0:Y%=&40+SN DIV 4:CALL(&FFF4)
```

loop through the word just spoken:

```
  240 REPEAT
```

energy is four bits; get more if necessary:

```
  250 IF BIT%<4 THEN PROCNextByte
  260 BIT%=BIT%-4:Energy = BYTE% DIV 2^BIT%:BYTE% = BYTE% MOD 2^BIT%
```

energy 0 means pause, no other data needed:

```
  270 IF Energy=0 THEN UNTIL FALSE
```

energy 15 means end of word:

```
  280 IF Energy=15 THEN UNTIL TRUE:SA=SA+1:UNTIL FALSE
```

repeat is one bit, pitch is six bits; if repeat, no more is needed:

```
  290 IF BIT%<1 THEN PROCNextByte
  300 BIT%=BIT%-1:Repeat = BYTE% DIV 2^BIT%:BYTE% = BYTE% MOD 2^BIT%
  310 IF BIT%<6 THEN PROCNextByte
  320 BIT%=BIT%-6:Pitch = BYTE% DIV 2^BIT%:BYTE% = BYTE% MOD 2^BIT%
  330 IF Repeat=1 THEN UNTIL FALSE
```

K1 to K4 are 18 bits total; if pitch is 0 no more is needed:

```
  340 PROCNextByte:PROCNextByte:IF BIT%<18 THEN PROCNextByte
  350 BIT%=BIT%-18:K1234 = BYTE% DIV 2^BIT%:BYTE% = BYTE% MOD 2^BIT%
  360 IF Pitch=0 THEN UNTIL FALSE
```

K5 to K10 are 21 bits total:

```
  370 PROCNextByte:PROCNextByte:IF BIT%<21 THEN PROCNextByte
  380 BIT%=BIT%-21:K5to10 = BYTE% DIV 2^BIT%:BYTE% = BYTE% MOD 2^BIT%
```

loop & end:

```
  390 UNTIL FALSE
  990 END
```

read the next byte from the speech processor:

```
 1000 DEF PROCNextByte
 1010 *FX 159,0,16
 1020 A%=158:R%=USR(&FFF4):B%=((R% DIV &10000)-1) AND &FF
 1030 BYTE%=BYTE%*&100+B%:BIT%=BIT%+8
 1035 SA=SA+1
 1040 ENDPROC
```

PHSAVE

PHSAVE creates a phrom file in *ROM format, which is useful to store programs which you can then access without any sideways-rom based filing system. The file is saved in the current filing system, so you need to use some other method to copy it to the root of a micro-SD card before loading it into Multiphrom.

OS1.2 only reads files from slots F0–FF, so it will warn you if you use slots C0–EF. The slot is only used to generate the filename, so you can move to a different slot simply by renaming the file. At the prompt, enter a filename to be read from the current filesystem and added to the rom, or a *command, which can be used to change directory (I'm not sure if changing filesystem will work), or #title, which creates an empty file to use as a catalog separator. When you have done all the files you want, press return at the prompt to complete the phrom file. This version does support files which run over into the next phrom (and changes the filename accordingly), but OS 1.2 gives Block? errors when you *CAT them, unless you turn on *OPT 1,2.

reserve space for OSFILE & OSGBPB operations, and define machine code routines:

```
1000 DIM BUFF% 260:DIM CTRL% 18:X%=CTRL% AND &FF:Y%=CTRL% DIV &100
1010 PROCCODE(BUFF%)
```

input slot number & ensure it's ok:

```
1020 REPEAT
1030 PRINT "Enter Phrom slot number (C0-FF)";:INPUT PHFN$
1040 PHFN = EVAL("&"+PHFN$)
1050 UNTIL PHFN >= &C0 AND PHFN <= &FF
1060 IF PHFN < &F0 THEN PRINT "Warning: OS 1.2 only reads files
from slots F0 - FF"
```

open the file for saving the created phrom:

```
1070 PROCOPEN
```

loop for each file saved:

```
1080 REPEAT
1090 PRINT "Slot " STR$~PHFN ", " &3FFF-EXT#(PHFILE) " bytes free."
1100 PRINT "Enter filename to save in Phrom, or *command, or
#title,":PRINT "or RETURN to finish";
1110 INPUT SFN$
```

if blank, end; if * pass to OSCLI; if # create empty file:

```
1120 IF SFN$ = "" THEN UNTIL TRUE:PROCCLOSE:END
1130 IF LEFT$(SFN$,1) = "*" THEN OSCLI MID$(SFN$,2):UNTIL FALSE
1140 IF LEFT$(SFN$,1)="#"THEN PROCTITLE(MID$(SFN$,2)):UNTIL FALSE
```

call OSFILE to read file info:

```
1150 !CTRL%=BUFF%:$BUFF%=SFN$
1160 A%=5:R%=USR(&FFDD)
1170 IF (R% AND &FF) <> 1 THEN PRINT "File not found.":UNTIL FALSE
```

L% is the length of the file, BLKS% is the number of 256-byte blocks, LA% is the load address, EA% the execution address, AT% the file attributes, SSFN$ the filename simplified for *ROM:

```
1180 L%=CTRL%!10:BLKS%=(L%-1) DIV 256 +1
1190 LA%=CTRL%!2:EA%=CTRL%!6:AT%=CTRL%?14
1200 SSFN$=SFN$:REPEAT:SSFN$=MID$(SSFN$,INSTR(SSFN$,".")+1):UNTIL
INSTR(SSFN$,".")=0
1210 SSFN$=LEFT$(SSFN$,10):N%=LEN(SSFN$)
```

calculate the address of the next file (or end of rom marker) in NXT%:

```
1220 NXT%=FNNXT(0)
```

open the file to be saved, and loop through the blocks:

```
1230 SFILE=OPENIN(SFN$)
1240 FOR I%=1 TO BLKS%
```

if we've reached the end of the rom, close the rom file and open a new one, recalculating the address of the next file:

```
1250 FLAG% = (PTR#PHFILE = NXT%):IF FLAG% THEN PROCCLOSE:
PHFN=PHFN+1: PROCOPEN: NXT%=FNNXT(I%-1)
```

if this is the first or last block of a file, or first block of a rom, write a full header, otherwise &23:

```
1260 IF FLAG% OR I%=1 OR I%=BLKS% THEN
PROCHEADER(I%=BLKS%,EOF#(SFILE),I%-1) ELSE BPUT#PHFILE,&23
```

read a block from the input file, calculate the CRC, and write both to the output file

```
1270 ?CTRL%=SFILE:CTRL%!1=BUFF%:CTRL%!5=256
1280 A%=4:R%=USR(&FFD1)
1290 BL%=CTRL%!1-BUFF%:!(CTRL%!1)=FNCRC(BL%)
1300 ?CTRL%=PHFILE:CTRL%!1=BUFF%:CTRL%!5=BL%+2
1310 A%=2:R%=USR(&FFD1)
```

loop, close the rom file and end:

```
1320 NEXT
1330 CLOSE#SFILE
1340 UNTIL FALSE
1350 END
```

calculate the next free address in the rom; if it's past the end reduce until it fits:

```
1360 DEF FNNXT(DONE%)
1370 LOCAL BASE%,FULL%
1380 BASE%=PTR#PHFILE+N%+20
1390 IF BASE% > &3FFE THEN = PTR#PHFILE
1400 FULL%=BASE%+L%+BLKS%*3-DONE%*259:IF BLKS%-DONE% > 1 THEN
FULL%=FULL%+N%+20
1410 IF FULL% <= &3FFE THEN = FULL% ELSE = BASE%+((&3FFE - BASE%)
DIV 259) * 259
```

close the rom file, first writing &2B at the end, and the end address in the header:

```
1420 DEF PROCCLOSE
1430 BPUT#PHFILE,&2B
1440 PTR#PHFILE=60:BPUT#PHFILE,FNREVBYTE(EXT#PHFILE AND
&FF):BPUT#PHFILE,FNREVBYTE(EXT#PHFILE DIV &100)
1450 CLOSE#(PHFILE)
1460 ENDPROC
```

open the rom file and write the (minimal) rom header:

```
1470 DEF PROCOPEN
1480 LOCAL I%
1490 PHFILE = OPENOUT("Ph_"+STR$~(PHFN)+"N")
1500 BPUT#PHFILE,0:BPUT#PHFILE,0:BPUT#PHFILE,ASC("("):
BPUT#PHFILE,ASC("C"):BPUT#PHFILE,ASC(")")
1510 FOR I%=1TO48:BPUT#PHFILE,32:NEXT
1520 BPUT#PHFILE,0:BPUT#PHFILE,0:BPUT#PHFILE,0:BPUT#PHFILE,0:
BPUT#PHFILE,0:BPUT#PHFILE,0:BPUT#PHFILE,0:BPUT#PHFILE,&FF:BPUT#PHFI
LE,&FF:BPUT#PHFILE,2:BPUT#PHFILE,0
1530 ENDPROC
```

write the full file header:

```
1540 DEF PROCHEADER(LAST%,EMPTY%,BN%)
1550 LOCAL I%
1560 $BUFF%=SSFN$
```

```
1570 BUFF%?N%=0:BUFF%!(N%+1)=LA%:BUFF%!(N%+5)=EA%:BUFF%!(N%+9)=BN%
1580 IF LAST% THEN BUFF%!(N%+11)=L% - PTR#SFILE:BUFF%?(N%+13)=&80
ELSE BUFF%!(N%+11)=256:BUFF%?(N%+13)=0
1590 IF EMPTY% THEN BUFF%?(N%+13)=BUFF%?(N%+13) OR &40
1600 IF AT% AND 4 THEN BUFF%?(N%+13)=BUFF%?(N%+13) OR 1
1610 BUFF%!(N%+14)=NXT%:BUFF%!(N%+18)=FNCRC(N%+18)
1620 BPUT#PHFILE,&2A:FOR I%=0 TO N%+19:BPUT#PHFILE,BUFF%?I%:NEXT
1630 ENDPROC
```

write an empty file:

```
1640 DEF PROCTITLE(N$)
1650 SSFN$=LEFT$(N$,10):N%=LEN(SSFN$)
1660 LA%=0:EA%=0:AT%=0:L%=0
1665 IF PTR#PHFILE >= &3FFE-N%-20 THEN PROCCLOSE:
PHFN=(PHFN+1)AND&FF:PROCOPEN
1670 NXT%=PTR#PHFILE+N%+21
1680 PROCHEADER(TRUE,TRUE,0)
1690 ENDPROC
```

call the machine code CRC routine:

```
1700 DEF FNCRC (A%)
1710 R%=USR(CRC)
1720 =!&81
```

call the machine code byte reverse:

```
1730 DEF FNREVBYTE (A%)
1740 R%=USR(REVERSE)
1750 =R% MOD 256
```

assemble the two machine code routines:

```
1760 DEF PROCCODE(BUFF%)
1770 DIM MC% 100
1780 FOR I=0 TO 2 STEP 2
1790 P% = MC%
1800 [OPTI
1810 .REVERSE STA &80
1820 ROL &80
1830 ROR A
1840 ROL &80
1850 ROR A
1860 ROL &80
1870 ROR A
1880 ROL &80
1890 ROR A
1900 ROL &80
1910 ROR A
1920 ROL &80
1930 ROR A
1940 ROL &80
1950 ROR A
1960 ROL &80
1970 ROR A
1980 RTS
1990 .CRC STA &80
2000 LDA #0
2010 STA &82
```

```
2020 STA &81
2030 TAY
2040 .NBYT LDA &81
2050 EOR BUFF%,Y
2060 STA &81
2070 LDX #8
2080 .LOOP LDA &81
2090 ROL A
2100 BCC B7Z
2110 LDA &81
2120 EOR #8
2130 STA &81
2140 LDA &82
2150 EOR #&10
2160 STA &82
2170 .B7Z ROL &82
2180 ROL &81
2190 DEX
2200 BNE LOOP
2210 INY
2220 CPY &80
2230 BNE NBYT
2240 RTS
2250 ]
2260 NEXT
2270 ENDPROC
```