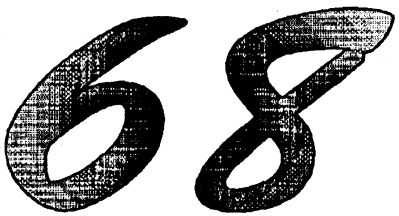

68 NEWS

Issue No. 11
November 1990

IN THIS ISSUE

68 News Asked to Stop Publishing

Just about the time we mailed out the previous issue of *68 News*, we learned that a new "68" magazine has been formed. It seems that this past summer, when we fell a few months behind schedule, some of our good friends took that to mean that *68 News* was dead. And so they decided to start another newsletter, tentatively called *The 68xxx Machines*. We have now been asked to stop publishing *68 News*, send *The 68xxx Machines* our subscription list, transfer our subscribers to them, and send them our articles and ads to be published there.

The 68xxx Machines will be a bit different from *68 News*. While their per-issue subscription rates will be somewhat lower than ours ($1 per issue for them, vs. $1.67 per issue for us), their ad rates are generally much higher ($75 for a half page in *The 68xxx Machines* vs. $10 for a half page in *68 News*). To get the advertisers to help pay for the costs, *The 68xxx Machines* will have to cover a hardware and software from many vendors. In other words, *The 68xxx Machines* would cover not just SK*DOS, but also perhaps OS-9, or Minix (a Unix clone) , as well as other hardware and software. The idea is to form a common magazine, replacing the many newsletters that individual vendors now send out themselves.

On the surface, I think this is a fine idea. The 68xxx world is small, and bringing it together under one roof can be beneficial to all. But I have several reservations.

When a magazine has to cover many different systems, the space devoted to any one has to be limited. There just may not be enough room for the material any one of us wants to see, and there may be much that we have no interest in.

Second, on a long-term basis, a magazine that relies heavily on advertisers must of necessity follow the majority of its readers. For example, some of the people involved with setting up the magazine are heavily into the Radio Shack Color Computer (CoCo). Since there are so many more CoCos than almost anything else, the magazine might turn into a CoCo magazine, for all I know.

I have seen cases where one vendor or group managed to take over a magazine and effectively downplay or even exclude other points of view. Whether it was preferential treatment from the publisher, or just a simple matter of one prolific vendor flooding the magazine with news releases, ads, articles, and photos, the ultimate effect is the same — a stilted, one-sided, unfair magazine. For example, I remember one vendor who wrote a monthly column (and even gave talks at meetings organized by the magazine), which turned out to plug all his own products and ignore everyone else's. It was designed to *look* impartial, but was in fact a free advertisement which made it look as though the magazine supported his products above those of other advertisers. I have no indication that this will happen with *The 68xxx Machines*, but I am afraid that it *might*.

As a result, I have decided to go with *The 68xxx Machines*'s suggestion on a trial basis. If the first issue comes out in January as expected, then you will receive an issue of *The 68xxx Machines* instead of *68 News*. Beyond that, things are less certain. I suspect that the ultimate solution may be to substitute *The 68xxx Machines* on an issue-for-issue basis, but still come out with an occasional issue of *68 News* to cover material which may not be appropriate for *The 68xxx Machines*, or for which they may simply not have room.

Let me know what you think.

Single-Drive Backup Program

by Mike Herman

Here is a useful program for those of us running SK*DOS on a system with just one floppy drive and too little memory for a RAMdisk. It is a backup program which allows making a copy of an entire disk on just one floppy drive. It copies as much of the disk into memory as fits, asks you to insert the destination disk, and copies the data out to the disk. If the memory is not large enough, then it asks you to swap disks several times, until the entire disk is copied.

```
* SDBACKUP - SINGLE DRIVE BACKUP FOR SK*DOS 68000
*
* ORIGINAL BY MIKE HERMAN, 9/90
* MODIFIED AND EXPANDED BY PETE STARK 11/17/90
* TO SAVE SPACE, WRITTEN FOR 256-BYTE SECTORS. IF SECTOR LENGTH
* IS CHANGED TO 512, CHANGE 352 TO 608 IN TWO PLACES BELOW
*
          LIB       SKEQUATE              LIBRARY FILE

          ORG       $0000                 START ADDR + OFFSET

SDBACKUP  BRA.S     START                 GOTO START
          DC.W      $0100                 VERSION NUMBER

START     LEA       MSG1(PC),A4           POINT TO INIT MESSAGE
          DC        PSTRNG                GO PRINT
          DC        GETNXT                READ DRIVE NUMBER
          SUB.B     #$30,D5               CONVERT FROM ASCII
          CMP.B     MAXDRV(A6),D5         CHECK IF VALID DRIVE
          BHI.L     HELP                  ON ERR XFER

* SET UP REGISTERS TO GO AHEAD

          MOVE.L    D5,D0                 SAVE DRIVE NUMBER
          MOVE.L    A6,A4                 POINT A4 TO USRFCB
          MOVE.B    D5,FCBDRV(A4)         INSERT DRIVE No INTO FCB
          MOVE.W    #$0003,FCBCTR(A4)     INSERT TR 0 SECT 3
          DC        SREAD                 READ SIS
          BNE.L     ERROR
          CLR.L     D1
          MOVE.W    FCBDAT+38(A4),D1      D1=LAST TRACK/SECT
          MOVE.L    MEMEND(A6),A0
          SUB.L     #610,A0               A0=END OF BUFFER
          CLR.L     D2
          MOVE.B    FCBPHY(A4),D7         CHECK IF FLOPPY
          AND.B     #$F0,D7
          CMP.B     #$10,D7
          BNE.L     NOTFLO                EXIT IF NOT FLOPPY

* REGISTER USAGE:
*  D0=DRIVE NUMBER   D1.W=LAST TRACK/SEC ON DISK
*  D2.W=LAST TRK/SEC READ   D3=READ FLAG      D4=EOF FLAG
*  A0=END OF BUFFER  A1=LAST BUFFER USED
```

```
* READ SECTOR DATA
*
READ    CLR.L   D3              CLEAR FLAG: NOTHING TO
WRITE
        CLR.L   D4              ALSO EOF FLAG
        LEA     ASKSRC(PC),A4   ASK FOR SOURCE DISK
        DC      PSTRNG          GO PRINT
        DC      GETCH           READ INPUT OF CR WHEN
READY
        LEA     FCBUFF(PC),A4   POINT TO BUFFER BEGINNING

RLOOP   ADD.B   #1,D2           NEXT SECTOR
        CMP.B   D1,D2           TIME TO GO TO NEXT TRACK?
        BLS.S   SECTOK          NO
        ADD.W   #$0100,D2       YES, NEXT TRACK
        MOVE.B  #1,D2           AND SECTOR 1
        CMP.W   D1,D2           PAST END OF DISK?
        BLS.S   SECTOK          NO
        MOVE.B  #$1,D4          YES, SET EOF FLAG
        BRA.S   WRITE           AND GO WRITE OUT ALL TO
DATE

SECTOK  MOVE.B  D0,FCBDRV(A4)   PUT DRV NO INTO FCB
        MOVE.W  D2,FCBCTR(A4)   AND TRACK/SECTOR
        DC      SREAD           READ SECTOR INTO BUFFER
        BNE.L   ERROR           EXIT ON ERROR
        MOVE.B  #1,D3           SET FLAG: SOMETHING TO
WRITE
        ADD.L   #352,A4         NEXT BUFFER AREA
        CMP.L   A4,A0           AT END OF BUFFER?
        BHI.S   RLOOP           NO, CONTINUE

*
* WRITE SECTOR DATA
*
WRITE   TST.B   D3              ANYTHING TO WRITE?
        BEQ.S   DONE            NO, SO EXIT
        MOVE.L  A4,A1           SAVE LAST BUFFER ADDRESS
        LEA     ASKDES(PC),A4   ASK FOR DESTINATION DISK
        DC      PSTRNG          GO PRINT
        DC      GETCH           READ INPUT OF CR WHEN
READY
        LEA     FCBUFF(PC),A4   POINT TO BUFFER BEGINNING

WLOOP   CMP.L   A4,A1           AT END OF BUFFER?
        BNE.S   WRITIT          NO, SO WRITE IT
        TST.B   D4              CHECK EOF FLAG
        BEQ.S   READ            READ MORE IF NOT EOF
        BRA.S   DONE            ELSE QUIT

WRITIT  DC      SWRITE          ELSE GO WRITE THE SECTOR
        BNE.S   ERROR           ERROR OUT
        ADD.L   #352,A4         THEN GO TO NEXT BUFFER
        BRA.S   WLOOP
```

```
* ALL DONE
*
DONE      DC        WARMST                RETURN TO SK*DOS
*
* ERRORS AND MESSAGES
*
HELP      LEA       HLPMSG(PC),A4
          DC        PSTRNG
          DC        WARMST
HLPMSG    DC.B      "SDBACKUP is a single-drive floppy disk
                       backup utility."
          DC.B      $D,$A
          DC.B      "The correct syntax is SDBACKUP ",$04

ERROR     DC        PCRLF
          MOVE.B    #$07,D4               BEEP
          DC        PUTCH
          DC        PERROR
          DC        WARMST

NOTFLO    DC        PCRLF
          LEA       NTFMSG(PC),A4
          DC        PSTRNG
          DC        WARMST

MSG1      DC.B      "SINGLE DRIVE BACKUP ",$04
ASKSRC    DC.B      "Insert source disk...CR when ready ",$04
ASKDES    DC.B      "Insert destination disk...CR when ready ",$04
NTFMSG    DC.B      "Specified drive is not a floppy disk.",4
*
          EVEN
FCBUFF    DS.B      1                     BEGINNING OF BUFFER

          END       SDBACKUP
```

Beginner's Corner

by Ron Anderson

I guess we ought to change the name of this to Assembler Corner, at least for the time being. Rather than forge on ahead very quickly into files and file handling, I thought perhaps we ought to spend a little more time on our program to add numbers. First, let's start with the program from last month and fix a deficiency. DECIN, as you will recall, won't accept negative numbers. Let's fix that problem by modifying our subroutine GETSTR. Since that routine is going to get more specific, I am going to rename it GETNUM. We'll fix it so that it looks to see if the first character it sees is a "-". If so, we'll tuck away that information and not put the minus sign in STRBUF. Then later, when we return from DECIN, we will check whether the value is negative and NEGATE the number if that is the case. It is easier to do than to describe in words.

Program ADD4 meets these requirements and brings up an example of something I mentioned previously but didn't explain at the time. I said that the

designers of the 68000 had placed a limitation on programmers. If a variable is tucked away along with the program access to it is limited. Look at the ADD4 listing below. The variable SIGN is declared at the end of the program. I can do this:

```
MOVE.B SIGN(PC),D0
```

but the following is illegal:

```
MOVE.B D0,SIGN(PC)
```

PC relative addressing can be used to read the value in a variable, but not to write a value to the variable. Instead we must go through two steps to use another addressing mode (We haven't formally talked about addressing modes yet).

```
LEA SIGN(PC),A0
MOVE.B D0,(A0)
```

This method will always work for writing to a memory location. Now I'll include the listing of ADD4. I've started with a fresh uncommented version and only commented the changes.

```
* ADD TWO NUMBERS INPUT BY USER
 NAM ADD4
*EQUATES

VPOINT EQU $A000
WARMST EQU $A01E
PSTRNG EQU $A035
PCRLF EQU $A034
OUT5D EQU $A038
DECIN EQU $A030
GETCH EQU $A029
LPOINT EQU 758

START DC VPOINT
 MOVE.L A6,A0
 LEA MSG1(PC),A4
 DC PSTRNG
 BSR.S GETNUM CHANGED SUBROUTINE
 LEA STRBUF(PC),A1
 MOVE.L A1,LPOINT(A0)
 DC DECIN
 MOVE.B SIGN(PC),D1 GET THE NEGATIVE SIGN FLAG
 BEQ.S ADD1 IF ZERO, NOT NEGATIVE
 NEG.W D5 OTHERWISE NEGATE THE NUMBER
ADD1 MOVE.W D5,D0
 LEA MSG2(PC),A4
 DC PSTRNG
 BSR.S GETNUM
 LEA STRBUF(PC),A1
 MOVE.L A1,LPOINT(A0)
 DC DECIN
 MOVE.B SIGN(PC),D1 SAME COMMENT AS ABOVE
 BEQ.S ADD2
 NEG.W D5
ADD2 ADD.W D5,D0
 LEA MSG3(PC),A4
 DC PSTRNG
```

```
 CLR.L D5
 MOVE.W D0,D4
 DC OUT5D
 DC WARMST

* GETNUM SUBROUTINE
GETNUM LEA STRBUF(PC),A1
 LEA SIGN(PC),A2 POINT AT SIGN VARIABLE
 CLR.B (A2) SET SIGN FALSE
GET1 DC GETCH
 CMP.B #'-',D5 SEE IF FIRST CHARACTER IS MINUS SIGN
 BNE.S GET2 IF NOT, ALL IS OK
 MOVE.B #$FF,(A2) IF SO, SET SIGN FLAG NON-ZERO
 BRA.S GET1  DON'T PUT "-" IN BUFFER
GET2 CMP.B #$20,D5 FROM HERE ON THERE ARE NO CHANGES.
 BEQ.S DONGET
 CMP.B #$0D,D5
 BEQ.S DONGET
 MOVE.B D5,(A1)+
 BRA.S GET1
DONGET MOVE.B #$0D,(A1)
 RTS

MSG1 DC.B "INPUT FIRST NUMBER  ",$04
MSG2 DC.B "INPUT SECOND NUMBER ",$04
MSG3 DC.B "SUM IS: ",$04
SIGN DC.B 1
STRBUF DS.B 30

 END START
```

Our program is growing. Note near the beginning of the program we have used MOVE.B SIGN(PC),D1. That instruction is followed by a BEQ.S. We haven't talked about the condition code register yet, but let me just say that simply moving a value to a data register causes the value to be tested and some condition codes set. The test for zero is done automatically and the condition code register ZERO FLAG is set appropriately so that the BEQ (branch if an equality test results in a zero value) works after simply moving the value to a register.

You might be thinking that we have done a lot of manipulation to get our "flag" set and cleared and stored away in the SIGN variable. You are absolutely correct. Since I've use D1 in the process, why not eliminate the variable SIGN and simply keep track of it in D1. Listing ADD5 follows, and it does just that.

```
* ADD TWO NUMBERS INPUT BY USER
 NAM ADD5
*EQUATES

VPOINT EQU $A000
WARMST EQU $A01E
PSTRNG EQU $A035
PCRLF EQU $A034
OUT5D EQU $A038
DECIN EQU $A030
GETCH EQU $A029
```

```
LPOINT EQU 758

START DC VPOINT
 MOVE.L A6,A0
 LEA MSG1(PC),A4
 DC PSTRNG
 BSR.S GETNUM
 LEA STRBUF(PC),A1
 MOVE.L A1,LPOINT(A0)
 DC DECIN
 TST.B D1 IF NOT ZERO, SIGN IS NEGATIVE
 BEQ.S ADD1
 NEG.W D5 NEGATE NUMBER IF NEGATIVE FLAG
ADD1 MOVE.W D5,D0
 LEA MSG2(PC),A4
 DC PSTRNG
 BSR.S GETNUM
 LEA STRBUF(PC),A1
 MOVE.L A1,LPOINT(A0)
 DC DECIN
 TST.B D1
 BEQ.S ADD2
 NEG.W D5
ADD2 ADD.W D5,D0
 LEA MSG3(PC),A4
 DC PSTRNG
 CLR.L D5
 MOVE.W D0,D4
 DC OUT5D
 DC WARMST

* GETNUM SUBROUTINE
GETNUM LEA STRBUF(PC),A1
 CLR.B D1 SET SIGN FALSE OR POSITIVE
GET1 DC GETCH
 CMP.B #'-',D5
 BNE.S GET2
 MOVE.B #$FF,D1 SET SIGN NEGATIVE
 BRA.S GET1  DON'T PUT "-" IN BUFFER
GET2 CMP.B #$20,D5
 BEQ.S DONGET
 CMP.B #$0D,D5
 BEQ.S DONGET
 MOVE.B D5,(A1)+
 BRA.S GET1
DONGET MOVE.B #$0D,(A1)
 RTS

MSG1 DC.B "INPUT FIRST NUMBER  ",$04
MSG2 DC.B "INPUT SECOND NUMBER ",$04
MSG3 DC.B "SUM IS: ",$04
STRBUF DS.B 30

 END START
```

Obviously the new version is simpler and has fewer instructions. Maybe not quite so obviously it is not as easy to follow for anyone but the author. A comment line:

```
* D1 is used to keep track of the sign of the input value
```

would surely help clarify the program. In fact, at the beginning of the main program and at the start of any subroutine, it would be a good idea to document the register usage. This is sometimes of great value in debugging a complex program. You can more easily find that you have used a register in a subroutine that was holding something important in the main program or a higher level subroutine.

Now that we have a program that will accept negative values, how about one more improvement. Let's make it so that it will accept numbers until you enter a zero and then print the total. Numbers may be positive or negative. The negative numbers are to be identified by preceding them with a "-" just as we have done so far. The only difference will be to change the prompts from "Enter First Number" and "Enter Second Number" to "Enter Number". We will do an unconditional branch back to get another number and add it until the number is zero and then we will print the total and exit. It will be smaller than our previous program:

```
* ADD NUMBERS INPUT BY USER UNTIL ZERO IS INPUT
 NAM ADD6
*EQUATES

VPOINT EQU $A000
WARMST EQU $A01E
PSTRNG EQU $A035
PCRLF EQU $A034
OUT5D EQU $A038
DECIN EQU $A030
GETCH EQU $A029
LPOINT EQU 758

START DC VPOINT GET POINTER TO SK*DOS VARIABLES
 MOVE.L A6,A0 SAVE POINTER TO VARIABLES
 CLR.W D0 TO HOLD SUM OF ENTRIES
ADD0 LEA MSG1(PC),A4
 DC PSTRNG
 BSR.S GETNUM
 LEA STRBUF(PC),A1
 MOVE.L A1,LPOINT(A0)
 DC DECIN
 TST.B D1 IF NOT ZERO, SIGN IS NEGATIVE
 BEQ.S ADD1
 NEG.W D5
ADD1 ADD.W D5,D0
 CMP.W #0,D5
 BNE.S ADD0 IF NOT ZERO GO GET MORE
 LEA MSG3(PC),A4
 DC PSTRNG
 CLR.L D5
 MOVE.W D0,D4
```

```
 DC OUT5D
 DC WARMST

* GETNUM SUBROUTINE
GETNUM LEA STRBUF(PC),A1
 CLR.B D1 SET SIGN FALSE OR POSITIVE
GET1 DC GETCH
 CMP.B #'-',D5
 BNE.S GET2
 MOVE.B #$FF,D1 SET SIGN NEGATIVE
 BRA.S GET1  DON'T PUT - IN BUFFER
GET2 CMP.B #$20,D5
 BEQ.S DONGET
 CMP.B #$0D,D5
 BEQ.S DONGET
 MOVE.B D5,(A1)+
 BRA.S GET1
DONGET MOVE.B #$0D,(A1)
 RTS
MSG1 DC.B "INPUT NUMBER  ",$04
MSG3 DC.B "SUM IS: ",$04
STRBUF DS.B 30

 END START
```

Make the changes in the previous program and assemble this one as ADD6. Give it a try. This one assembled correctly on the first try and did what I expected it to do. Now you can balance your checkbook provided your balance never exceeds $327.67 nor is in the red by more than $327.68. (Just omit the decimal point in your entries). I'll leave it as the first exercise for you to work independently, to change the .W arithmetic to .L arithmetic. I believe DECIN will already work for Longs, but you will have to use OUT10D rather than OUT5D in order to print out the longer number. When you get done, you can balance your checkbook if the numbers are less than about $20,000,000. If that is a problem for you, let me know and we will do a double precision long integer version that can handle the National Debt!

On a different subject, it might be time to introduce a few more quirks in the 68000 instruction set (hereafter when I say 68000 I mean the entire family, it gets tiresome to write 68XXX all the time). There is a special short instruction that may be used to add a small amount to a register. Edit the following two line program and assemble it using ASM:

```
ADD.W #1,D0
ADDQ.W #1,D0
END
```

You will get the following listing:

```
00002 00000000 06450001                          add.w    #1,d5
00003 00000004 5245                              addq.w   #1,d5
00004 00000006                                   end
```

Notice that the second line generates a shorter instruction (i.e. hex code) than the first. In fact the first is a 4 byte instruction and the second is only 2. The Q

(for Quick) version of Add and Subtract work for immediate values from 1 to 8. There were three bits left over in the instruction (speaking loosely) so the people who worked out the instruction set coded the immediate value right into the instruction. In the case of the regular ADD instruction the amount to be added is coded in the second word of the instruction (the 0001 in the above listing). Again, I will refer you to the 68000 user's manual or your book on assembler programming on the 68000. The situation here is similar to the one that we discussed in considering the branch instructions, BRA vs. BRA.S. The shorter form of the instruction is more limited in scope but it reduces the size of the object code and it runs faster, and so should be considered if speed is critical. With a megabyte of memory available, you might well ask why a programmer should worry about saving two bytes here and there, and the question is valid. Back in the days when a computer had 4K of memory, programmers were very concerned about saving a few bytes, and some of this is a holdover from those days. Obviously with all that memory, space is no longer as important a consideration, but speed might be a larger one.

Generally compilers ignore these shortcut instructions and use the BRA form and the ADD form without exception. This is a case of simplifying the compiler at the expense of a few more bytes of code. I ought to mention that some assemblers automatically choose the correct instruction. ASMK from Palm Beach Software, for example, will generate an ADDQ machine code wherever an ADD has a value within proper range. ASMK won't accept an ADDQ instruction, however. It wants to do it automatically and it balks with an error message if you try to tell it to use the ADDQ. ASMK does something else that is rather nice. If you use a BRA.. instruction, and it is within the range of a BRA.S, it will flag the instruction so you can change it. In this case, the assembler doesn't do it automatically, but it does tell you that you may do it. In a long program you may find that when you change all the flagged branch instructions to short branches, a few more will be brought within range, so the process is iterative. Perhaps this is why the assembler doesn't do it automatically.

Our program from above yields the following when assembled with ASMK:

```
000000 5245                     add.w    #1,d5
000002 4E71 4E71 4E71           addq.w   #1,d5
***  UNRECOGNIZABLE MNEMONIC OR MACRO
                                end
```

It automatically generates the machine code for ADDQ on the ADD.W instruction, and it complains when you try to tell it to use ADDQ.

If you look through the user's guide you will find a couple other versions of ADD. Both of the assemblers mentioned here handle those automatically. In the original instruction set we would have to use ADDI #32,D7. Both assemblers see the immediate sign (#) and generate the ADDI instruction. ADDA is also mentioned for use in adding to an address register. This is also handled by both assemblers. There are a few more subtle differences in the two assemblers that we needn't worry about for the present.

Well, next time we will get into reading from and writing to disk files, I promise. Perhaps at that point, you will be able to read and understand books on the subject without help.

I truly hope these four articles have helped you to get started. Somehow it is difficult to get through that first step of firing up the computer and getting it to do something. Welcome to the world of computer programmers as opposed to being appliance users. Future columns will deal with using compilers and interpreters, and reviews of software that you can use.

Print Spooling on the PT68K-2

by Dr. Michael Randall

(Editor's Note: Mike sent us the enclosed code for inclusion in the 68 News, *but did not enclose an article with it. I have therefore added the following paragraphs by way of explanation.)*

A *print spooler* lets you print a file at the same time as you do something else; the word *spooler* dates back to the early days of large computers, when the file was written to a spool of tape, to be printed at a later time. For example, a print spooler allows you to print out an assembly listing at the same time as you edit or assemble another file.

Without a print spooler, you would have to wait until the printing is finished before you could edit or assemble the next file; the spooler allows you to do both at the same time. The computer can handle both jobs easily because printing keeps the printer busy, but involves relatively little work for the CPU.

The print spooler is nothing more than a program (or actually, two programs in our case), which sets up a timer (the DUART in our case) to generate periodic interrupts. Each time this interrupt arrives, the CPU stops the current program (which may be an editor, assembler, or any other program you might be running), temporarily goes off to send the next character to the printer, and then resumes your program as if nothing had happened. Since the interruption is very short, you will generally never notice that the CPU went off to do something else for a moment.

To use Mike's print spooler, you must do three things:

Step 1. Prepare the file or files you want to print. For most normal text files, the file may already exist as .TXT files, in which case you need do nothing. In some cases, you may have to go out of your way to prepare the file. For example, to spool an assembly listing, you will have to tell the assembler to write a listing file, rather than print it directly. This could be done by redirecting its output to a file. For example, the command

```
ASM TESTFILE -B 4.LISTFILE.OUT
```

would tell the assembler to assemble TESTFILE, not generate a binary file, and send all output to a file called LISTFILE.OUT on drive 4. (On my system, drive 4 is generally a RAMdisk, so this would be a strictly temporary file.)

Step 2. Use the DEVICE command to load the PARSPOOL printer driver and call it PSPL. The following command would do the job:

```
DEVICE PARSPOOL AT 3 AS PSPL
```

Note that a different driver is needed; the normal PARALLEL driver you may have been using to date is not set up for spooling.

Step 3. Use the PRINT command to send the file to the driver. In our example, the command would be

```
PRINT LISTFILE
```

PRINT is used here instead of LIST. Many of you have been using a command such as LIST PRTR to send a listing to a PRTR driver; now you would use PRINT instead. The PRINT command defaults to an extension of .OUT, so we did not need to add an .OUT after LISTFILE; for other extensions, you would need to add it.

The PARSPOOL driver code follows:

```
          NAM PARSPOOL.TXT
          OPT PAG
          LIB SKEQUATE.TXT
          PAG

* PARALLEL DEVICE DRIVER FOR SK*DOS/68K
* THIS VERSION IS CONFIGURED FOR THE PT-68K-2 COMPUTER
* IT IMPLEMENTS PRINTER SPOOLING USING DUART 2 TO                +
* GENERATE TIMER INTERRUPTS                                      +
* OCNTRL $0081 GIVES THE CALLER ACCESS TO THE SPOOL              +
* QUEUE DATA STRUCTURE                                           +
* IT IS BASED ON PETER STARK'S "PARALLEL.TXT"                    +
* PARTS ADDED ARE MARKED "+" PARTS CHANGED ARE MARKED "X"        +

* THE FORMAT OF A DEVICE DRIVER FOR SK*DOS IS VERY RIGID,
* ESPECIALLY AT THE BEGINNING AND VERY END. YOU MAY
* SUBSTITUTE YOUR OWN DRIVER CODE, BUT MAKE SURE TO USE THE SAME
* FORMAT FOR THE DRIVER AS THIS EXAMPLE. NOTE ESPECIALLY THAT
* THIS DRIVER IS NOT EXECUTABLE; IT IS TO BE LOADED INTO MEMORY
* BY THE 'DEVICE' UTILITY, AND MUST BE POSITION-INDEPENDENT.

**************** IMPORTANT !! ****************************
** IF USING ASM.COM TO ASSEMBLE THIS FILE, CHANGE "PAG"
** TO "PAGE" IN LINES 2 AND 3, AND MAKE SURE TO
** USE THE -F OPTION
**********************************************************

**********************************************************
* PART 1. BEGINNING, VERSION NUMBER, AND A 'DN' MARKER
* WHICH IS CHECKED BY 'DEVICE' TO AVOID ERRORS.
**********************************************************

START     BRA.S START              NEVER EXECUTED

          DC.W $0001               VERSION NUMBER
          DC.W $444E               'DN' ID MARKER

**********************************************************
* PART 2. LENGTH SPECIFICATION. THE NEXT LONG WORD DEFINES
* THE LENGTH OF THE DRIVER TO DEVICE.COM. THEEND IS A
* LABEL WHICH IS PLACED AT THE VERY END OF THE DRIVER
```

```
*************************************************************
LENGTH    DC.L THEEND

*************************************************************
* PART 3. ENTRY POINT POINTERS. THE FOLLOWING POINTERS
* DEFINE THE ENTRY POINTS INTO THE DRIVER. ALL ARE
* RELATIVE TO THE ORIGIN
*************************************************************

          DC.L INIPO1              DRIVER INITIALIZATION      X
          DC.L INSTAT              INPUT STATUS
          DC.L INCHAR              INPUT CHARACTER WITH ECHO
          DC.L INCHAN              INPUT CHAR WITHOUT ECHO
          DC.L ICNTRL              INPUT CONTROL ENTRY
          DC.L OUSTA1              OUTPUT STATUS              X
          DC.L OUCHR1              OUTPUT A CHARACTER         X
          DC.L OCNTR1              OUTPUT CONTROL ENTRY       X
          DC.L INSTA1              INPUT STATUS (1 CHAR ONLY)
          DC.L INCHN1              INPUT 1 CHAR ONLY, NO ECHO
          DC.L BFLUSH              FLUSH TYPEAHEAD BUFFER

*************************************************************
* PART 4. THE FOLLOWING LINE DEFINES THE BEGINNING OF THE ACTUAL
* CODE AS BEING AT $0000. IT SERVES TO BREAK UP THE OBJECT FILE
* TO MAKE IT EASIER FOR 'DEVICE' TO LOAD INTO THE CORRECT PLACE.
*************************************************************

          ORG $0000

*************************************************************
* PART 5. DATA AREA USED BY THE DEVICE DRIVER FOR VARIOUS
* PURPOSES.
*************************************************************

* BYTES 0-12:
* NAME OF THIS DEVICE DRIVER DISK FILE - 12 CHARACTERS PLUS 04
* 'PRTSPOOL' IS REQUIRED BY PRINT.COM WHICH SEARCHES THE      +
* DEVICE TABLE FOR A DEVICE WITH THIS NAME                    +

DRNAME    DC.B 'PRTSPOOL.DVR',4                               X

* BYTE 13:
* DEVICE NUMBER SO THIS DRIVER KNOWS WHICH DEVICE IT IS
DEVNUM    DC.B 0                   WILL BE FILLED IN BY 'DEVICE'

* BYTES 14-17:
* ADDRESS OF DEVICE DESCRIPTOR FOR THIS DRIVER
DEVADD    DC.L 0                   WILL BE FILLED IN BY 'DEVICE'

*************************************************************
* PART 6. DEVICE INITIALIZATION. THIS CODE INITIALIZES PORT A
* OF THE 68230 PI/T AT $FE0080 IN THE PT68K-2 COMPUTER
* AND TIMER INTERRUPTS USING DUART 2. IT IS                   +
* ONLY EXECUTED WHEN LOADED BY 'DEVICE'. ALL REGISTERS CAN BE
* CHANGED.
```

```
*****************************************************************

* HARDWARE EQUATES FOR THE PORTS

PIT       EQU $FE0081              PIT ADDRESS
GENCON    EQU PIT                  GENERAL CONTROL REG
PADIR     EQU PIT+4                DIRECTION REG
PACONT    EQU PIT+12               CONTROL REGISTER
DATREG    EQU PIT+16               DATA REGISTER
STAREG    EQU PIT+26               PORT STATUS REGISTER

BASE EQU $FE0040                   DUART2 BASE ADDRESS        +
ACR EQU BASE+$9                                               +
ISR EQU BASE+$B                                               +
IMR EQU BASE+$B                                               +
CTUR EQU BASE+$D                                              +
CTLR EQU BASE+$F                                              +
STARTC EQU BASE+$1D                                           +
STOPC EQU BASE+$1F                                            +
VEC5 EQU $74                  LEVEL 5 AUTOINTERRUPT VECTOR    +
DRVUSD EQU $113C                                              +
SECTRD EQU $1020                                              +
DIREAD EQU $110C                                              +

* INITIALISE TIMER                                            +
INIPO1    DC INTDIS            DISABLE INTERRUPTS             +
          LEA INT(PC),A0                                      +
          LEA SAVVEC(PC),A1                                   +
          MOVE.L VEC5,(A1)     SAVE NORMAL VECTOR             +
          MOVE.L A0,VEC5       INSTALL NEW VECTOR             +
          MOVE.B #$F0,ACR      TIMER MODE SET UP              +
          MOVE.B #$4,CTUR                                     +
          MOVE.B #$84,CTLR     10ms INT                       +
          TST.B STOPC          CLEAR ISR[3]                   +
          TST.B STARTC         START TIMER CLEAN              +
          MOVE.B #8,IMR        ENABLE TIMER INTERRUPT         +
          BSR.S INIPOR         PRINTER INIT                   +
          DC INTENA            ENABLE INTERRUPTS              +
          RTS                                                 +

*   INITIALISE PRINTER PORT A FOR UNIDIRECTIONAL OUTPUT,
*   H2 PULSED OUTPUT MODE, H1 TO DETECT RISING EDGE

INIPOR    MOVE.B #0,GENCON          START WITH GENERAL CTRL =0
          MOVE.B #$FF,PADIR         8 OUTPUT BITS
          MOVE.B #$78,PACONT        PORT A,SUBMODE 01,PULSED
          MOVE.B #$10,GENCON        ENABLE PORT
          RTS

*****************************************************************
* PART 7. INPUT PORT STATUS CHECK. THIS ROUTINE IS TO RETURN
* TWO THINGS:
*    1. ZERO IF NO CHARACTER IS READY, NON-ZERO OTHERWISE
*    2. D5=0 IF NO CHARACTER IS READY, ELSE THE NUMBER OF
*       CHARACTERS READY. IN NON-INTERRUPT SYSTEMS, D5 SHOULD
*       RETURN A 1; ONLY IN INTERRUPT-DRIVEN SYSTEMS WILL IT
```

```
*       INDICATE A REAL NUMBER OF CHARACTERS IN INPUT BUFFER
* PRESERVE ALL REGISTERS
* ON THE PRINTER PORT, HOWEVER, THERE IS NO INPUT SO ZERO
* NOTE CHANGES INTRODUCED IN VERSION 0004 :
*   THERE ARE NOW TWO INSTA- ENTRIES, ALTHOUGH BOTH DEFAULT
* TO THE SAME ROUTINE IF THERE IS NO TYPEAHEAD BUFFER ON THE
* INPUT PORT: INSTAT CHECKS TO SEE WHETHER THERE IS ANY CHARACTER
* IN THE TYPEAHEAD BUFFER, WHEREAS INSTA1 CHECKS ONLY TO SEE
* WHETHER THERE IS A 'LAST' CHARACTER AT THE END OF THE BUFFER
* WHICH HAS NOT YET BEEN INPUT WITH THE INCHN1 ENTRY
*****************************************************************

INSTAT    MOVE.B #0,D5              NO CHARACTER READY
          RTS
INSTA1    EQU INSTAT

*****************************************************************
* PART 8. GET INPUT CHARACTER FROM PORT INTO D5 AND ECHO IT TO
* THE OUTPUT PORT. IF NO CHARACTER IS READY, WAIT FOR IT.
* PRESERVE THE PARITY BIT, AND PRESERVE ALL REGISTERS
* ON THE PRINTER PORT, HOWEVER, THERE IS NO INPUT SO ZERO
* THIS ENTRY USES THE TYPEAHEAD BUFFER, IF ANY
*****************************************************************

INCHAR    MOVE.B #0,D5              RETURN NOTHING
          RTS

*****************************************************************
* PART 9. GET INPUT CHARACTER FROM PORT INTO D5 WITHOUT ECHOING
TO
* THE OUTPUT PORT. IF NO CHARACTER IS READY, WAIT FOR IT.
* PRESERVE THE PARITY BIT, AND PRESERVE ALL REGISTERS
* ON THE PRINTER PORT, HOWEVER, THERE IS NO INPUT SO ZERO
* NOTE CHANGES INTRODUCED IN VERSION 0004 :
*   THERE ARE NOW TWO INCH-- ENTRIES, ALTHOUGH BOTH DEFAULT
* TO THE SAME ROUTINE IF THERE IS NO TYPEAHEAD BUFFER ON THE
* INPUT PORT: INCHAN TAKES THE NEXT CHARACTER FROM THE TYPEAHEAD
* BUFFER (AND CLEARS THE INSTA1 FLAG), WHEREAS INCHN1 TAKES ONLY
* THE CHARACTER FROM THE END OF THE TYPEAHEAD BUFFER, AND CLEARS
* BOTH FLAGS.

*****************************************************************

INCHAN    MOVE.B #0,D5              RETURN NOTHING
          RTS
INCHN1    EQU INCHAN

*****************************************************************
* PART 10. INPUT CHANNEL CONTROL. THIS DRIVER DOES NOT
* IMPLEMENT INPUT CONTROL, SO SIGNAL ERROR AND RTS
*****************************************************************

ICNTRL    AND.B #$FB,CCR            RETURN NON-ZERO ERROR
          RTS

*****************************************************************
```

```
* PART 11. OUTPUT STATUS. RETURN ZERO IF OUTPUT IS NOT
* READY, NON-ZERO IF READY TO OUTPUT NEXT. PRESERVE ALL
* REGISTERS. IF NO HANDSHAKING IS USED, OR IF HARDWARE
* HANDSHAKING IS USED, THEN SIMPLY CHECK BUSY BIT.
************************************************************

OUSTA1    MOVE.L A0,-(A7)          SAVE                      +
          LEA SPAF(PC),A0                                    +
          TST.B (A0)                                         +
          BPL.S OCERR              IF SPOOL NOT IDLE         +
          BTST #0,STAREG           = BSR OUSTAT              +
OSX       MOVE.L (A7)+,A0          RESTORE                   +
          RTS                                                +

************************************************************
* PART 12. OUTPUT CHARACTER FROM D5 TO OUTPUT PORT. IF NOT
* READY, JUST WAIT FOR IT. PRESERVE ALL REGISTERS (INCLUDING D5)
************************************************************

OUCHAR    BTST.B #0,STAREG           CHECK IF READY
          BEQ.S OUCHAR              WAIT UNTIL READY
          MOVE.B D5,DATREG          OUTPUT THE CHARACTER
          RTS

OUCHR1    MOVE.L A0,-(A7)       SAVE                         +
          LEA SPAF(PC),A0                                    +
          TST.B (A0)            ZERO IF SPOOLING ACTIVE      +
          BPL.S OCERR                                        +
          BSR.S OUCHAR          PRINT IF IDLE                +
          MOVE.L (A7)+,A0       RESTORE                      +
          RTS                                                +

OCERR     MOVE.W #$FFF1,D4                                   +
          DC $A032 OCNTR1       REDIRECT TO ERROR OUTPUT     +
          LEA SPMSG(PC),A4                                   +
          DC PSTRNG                                          +
          DC WARMST             ERROR IF SPOOLING NOT IDLE   +

************************************************************
* PART 13. OUTPUT CHANNEL CONTROL.
************************************************************
* $0081 USED BY PRINT.COM                                    +
OCNTR1    CMP.W #$81,D4                                      +
          BEQ.S OCNT81                                       +

OCNTRL    AND.B #$FB,CCR             RETURN NON-ZERO ERROR
          RTS

* RETURN PTR TO QUEUE DATA STRUCTURE IN D5.L                 +
OCNT81    MOVE.L A0,-(A7)       SAVE A0                      +
          LEA AQP(PC),A0                                     +
          MOVE.L A0,D5                                       +
IEXIT     MOVE.L (A7)+,A0       RESTORE A0                   +
          OR.B #4,CCR           SET Z - NO ERROR             +
          RTS                                                +
```

```
*********************************************************
* PART 14.                    WHEN A TYPEAHEAD BUFFER
* EXISTS, THIS FLUSHES IT AND CLEARS BOTH INSTAT AND
* INSTA1 FLAGS TO INDICATE THAT ALL IS EMPTY.
*********************************************************

BFLUSH    RTS                          IN THIS CASE DOES NOTHING

*=========================================================

* INTERRUPT ROUTINE FOR TIMER INTERRUPTS                          +
INT       BTST #3,ISR                                             +
          BNE.S TIMINT                                            +
          MOVE.L SAVVEC(PC),-(A7) NORMAL INT ADDR ON STACK        +
          RTS GO TO NORMAL INT ROUTINE IF NOT TIMER INT           +
TIMINT    TST.B STOPC             TO CLEAR ISR[3]                 +
          MOVE.L A0,-(A7)         SAVE A0 ON STACK                +
          LEA SPAF(PC),A0                                         +
          TST.B (A0)                                              +
          BNE.S SPX1              IGNORE IF IDLE OR SUSPENDED     +
          BTST.B #0,STAREG                                        +
          BEQ.S SPX1              IGNORE IF PRINTER NOT READY     +
          MOVEM.L D0-D7/A1-A6,-(A7) SAVE REGISTERS                +
*SPOOL OPERATIONS                                                 +
          LEA SPFCB(PC),A4                                        +
          BSR TREAD                                               +
          BEQ.S EOF                                               +
          TST.B D5                                                +
          BEQ.S SPEXIT            IGNORE NULLS                    +
          MOVE.B D5,DATREG        PRINT CHAR = BSR OUCHAR         +
          BRA.S SPEXIT                                            +
EOF       LEA SPAF(PC),A2         A2=SPAF                         +
          MOVE.B #1,(A2)          SUSPEND                         +
* on end-of-file or file error print form-feed                    +
* and go to next file if any                                      +
          MOVE.B #$C,D5                                           +
          BSR OUCHAR              PRINT FF                        +
          LEA AQP(PC),A0                                          +
          MOVE.L (A0),A0                                          +
          MOVE.B #0,(A0)          FREE THE FILE DEF ENTRY         +
          BSR BUMP                BUMP POINTERS & OPEN FILE IF ANY+

SPEXIT    MOVEM.L (A7)+,D0-D7/A1-A6 RESTORE REGS                  +
SPX1      MOVE.L (A7)+,A0         RESTORE A0                      +
          RTE                                                     +

* BUMP POINTERS                                                   +
BUMP      LEA AQP(PC),A0                                          +
          MOVE.W #9,D5            MOVE 10 POINTERS UP             +
SS2       MOVE.L 4(A0),(A0)+                                      +
          DBRA D5,SS2                                             +
          MOVE.L #0,(A0)          CLEAR LAST POINTER              +
          BSR OPFI                                                +
          TST.B (A2)                                              +
          BMI.S SS3               IF IDLE (SET BY OPFI IF Q EMPTY)+
          MOVE.B #0,(A2)          SET ACTIVE IF SUSPENDED         +
```

```
SS3         RTS                                                    +

* OPEN FILE IF NRINQ # 0                                          +
OPFI        LEA NRINQ(PC),A0                                       +
            TST.B (A0)                                             +
            BNE.S OF1                                              +
            MOVE.B #$FF,(A2)      SET FLAG TO INDICATE SPOOL IDLE+
            RTS                                                    +
OF1         SUB.B #1,(A0)         ADJUST NR IN Q                   +
            LEA AQP(PC),A0                                         +
            MOVE.L (A0),A0        A0=FILE DEF STRING               +
            LEA SPFCB(PC),A4                                       +
            MOVE.B 1(A0),FCBDRV(A4)                                +
            MOVE.W 14(A0),FCBCTR(A4)                               +
            LEA COUNT(PC),A0                                       +
            MOVE.B #255,(A0)                                       +
            RTS                                                    +
FCBPTR      EQU 80                UNUSED SPACE IN FCB              +
TREAD       LEA COUNT(PC),A0                                       +
            CMP.B #255,(A0)                                        +
            BNE.S NXTBYT                                           +
* READ NEXT SECTOR                                                +
            BSR S1READ                                             +
            BNE TRERR                                              +
            MOVE.W FCBDAT(A4),FCBCTR(A4) LINK TO NEXT              +
            MOVE.B #252,(A0)      REDUCE COUNT                     +
            LEA 100(A4),A3        POINT TO 1ST DATA BYTE           +
            BRA.S NB1                                              +
NXTBYT      MOVE.L FCBPTR(A4),A3 GET DATA PTR                      +
NB1         MOVE.B (A3)+,D5                                        +
            MOVE.L A3,FCBPTR(A4) UPDATE DATA PTR                   +
            SUB.B #1,(A0) 0       WHEN LAST BYTE READ              +
            BEQ.S LSTBYT          LAST BYTE READ                   +
            RTS                                                    +
LSTBYT      TST.W FCBCTR(A4)                                       +
            BEQ.S ITSEOF          THAT WAS LAST SEC RETN WITH Z SET+
            MOVE.B #255,(A0)      Z NOW CLEAR                      +
            RTS                                                    +
ITSEOF      MOVE.B #8,FCBERR(A4)                                   +
TRERR       OR.B #4,CCR           SET Z FOR EOF OR READ ERR        +
            RTS                                                    +

* SINGLE SECTOR READ - DON'T USE DC SREAD !!!                      +
S1READ      MOVE.B 3(A4),D5                                        +
            AND.L #$F,D5                                           +
            MOVE.L #DRVUSD,A3                                      +
            MOVE.B 0(A3,D5),FCBPHY(A4)                             +
            JSR SECTRD                                             +
            JMP DIREAD                                             +

*------------------------------------------------------------------ +
* DATA AREA                                                        +
*------------------------------------------------------------------ +
SAVVEC      DC.L 0                SAVE OLD VECTOR HERE             +
NRINQ       DC.B 0                NR FILES IN Q                    +
SPAF        DC.B $FF              = SPOOLING IDLE,                 +
*                                 1= SUSPENDED, 0 = ACTIVE         +
```

```
AQP       DC.L 0                ACTIVE QUEUE POINTER          +
QP0       DC.L 0,0,0,0,0,0,0,0,0,0 TEN Q POINTERS             +
FS0       DC.L 0,0,0,0          1ST FILE DEFINITION STRING    +
          DC.L 0,0,0,0                                        +
          DC.L 0,0,0,0                                        +
          DC.L 0,0,0,0                                        +
          DC.L 0,0,0,0                                        +
          DC.L 0,0,0,0                                        +
          DC.L 0,0,0,0                                        +
          DC.L 0,0,0,0                                        +
          DC.L 0,0,0,0                                        +
          DC.L 0,0,0,0          TEN FILE DEF STRS             +
SPFCB     EQU *                 SPOOLER FCB                   +
          RPT 38                X16=608                       +
          DC.L 0,0,0,0
COUNT     DC.B 0                                              +
SPMSG     FCC /PRINT ERROR - SPOOLING ACTIVE/,4               +
*********************************************************
* THE END. LABEL THEEND IS USED TO CALCULATE LENGTH OF
* DRIVER. NOTE THAT THERE IS NO TRANSFER ADDRESS.
*********************************************************

THEEND    EQU *
          END
```

The following is the PRINT.COM program:

```
          NAM       PRINT.COM
          OPT       PAG
          PAG
          LIB       SKEQUATE.TXT
*-----------------------------------------------------------
* PRINT.COM - ENTER FILE IN SPOOL PRINT QUEUE
*-----------------------------------------------------------
OCNTRL    EQU       $A032
PSPL      EQU       'PSPL'
* OFFSETS TO SPOOL DATA STRUCTURE
NRINQ     EQU       -2
SPAF      EQU       -1
AQP       EQU       0
QP0       EQU       4
FS0       EQU       $2C
SPFCB     EQU       $CC
COUNT     EQU       $32C

START     MOVE.L    #0,D4
          DC        OCNTRL          GET CURRENT DEVICE IN D5
          AND.L     #7,D5
          OR.W      #$FFF0,D5
          MOVE.L    D5,D2           OCNTRL WORD FOR CURRENT DEVICE
IN D2
* CHECK FOR PSPL DEVICE
          DC        VPOINT
          MOVE.L    A6,A4           USE USRFCB
* search the device table for a device named PRTSPOOL
          LEA       SPNAME(PC),A0   POINT TO NAME TO BE FOUND
          LEA       DEVTAB(A6),A1   POINT TO DEVICE TABLE 1ST ENTRY
```

```
        CLR.L   D1              START AT DEVICE 0
SRCH    MOVE.L  4(A1),A2        POINT TO START OF DRIVER CODE
        MOVE.L  (A2),D0
        CMP.L   (A0),D0
        BNE.S   NOTIT
        MOVE.L  4(A2),D0
        CMP.L   4(A0),D0
        BNE.S   NOTIT
* WE HAVE FOUND THE DRIVER (D1)
        BRA.S   NEWDEV
NOTIT   ADD.B   #1,D1
        CMP.B   #8,D1
        BEQ.S   NOTFND          WE COULD'T FIND IT
        ADD.L   #80,A1          NEXT ENTRY IN DEVICE TABLE
        BRA.S   SRCH
* REPORT DEVICE NOT INSTALLED
NOTFND  LEA     NOPSPL(PC),A4
        DC      PSTRNG
        DC      WARMST
* SAVE OCNTRL WORD FOR PSPL DEVICE IN D1
NEWDEV  OR.W    #$FFF0,D1
* GET FILE SPEC INTO FCB
        DC      GETNAM
        BCC.S   EXTEN
* REPORT IMPROPER FILE SPEC - TELL USER THE RIGHT SYNTAX
        LEA     BADFIL(PC),A4
        DC      PSTRNG
        DC      WARMST
EXTEN   MOVE.L  #11,D4
        DC      DEFEXT          (.OUT)
* SWITCH TO PSPL DEVICE
        MOVE.L  D1,D4
        DC      OCNTRL
        MOVE.W  #$81,D4
        DC      OCNTRL          GET PTR TO SPOOL DATA IN D5
        MOVE.L  D5,A3           = SPOOL DATA STRUCTURE
        MOVE.L  D2,D4
        DC      OCNTRL          SWITCH TO FORMER DEVICE
        MOVE.B  SPAF(A3),D3     CURRENT SPAF SAVED IN D3
        BMI.S   I1
        MOVE.B  #1,SPAF(A3)     IF NOT IDLE, SUSPEND
I1      CMP.B   #10,NRINQ(A3)
        BEQ     FULL            NO FREE ENTRIES
* OPEN AND CLOSE THE FILE
        DC      FOPENR
        BNE     ERROR
        DC      FCLOSE
        BNE     ERROR
* GET FREE ENTRY IN FILE DEF TABLE
        LEA     FS0(A3),A0
I2      TST.B   (A0)            LOOK FOR FREE ENTRY
        BEQ.S   I3
        ADD.L   #16,A0          NEXT ENTRY
        BRA.S   I2
I3      MOVE.L  A0,A2           A2,A0=FILE DEF ENTRY
        MOVE.B  #$20,(A0)+      SPACE (NON-ZERO)
```

```
        LEA     3(A4),A1        A1=FCB FILE DEF
        MOVE.W  #11,D0          12 BYTES TO COPY
CPY     MOVE.B  (A1)+,(A0)+
        DBRA    D0,CPY          COPY FCB TO FILE DEF

* ENTER FILE DEF IN Q
        MOVE.L  #0,D5
        MOVE.B  NRINQ(A3),D5
        ASL.B   #2,D5           X4
        LEA     QP0(A3),A1
        ADD.L   D5,A1           (A1) IS NEXT Q PTR
        MOVE.L  A2,(A1)         STORE FILE DEF ADDR
        ADD.B   #1,NRINQ(A3)    UPDATE
        MOVE.W  FCBFTR(A4),14(A2) 1ST TRK/SEC TO FILE DEF

* IF NOT IDLE RESTORE SPOOL STATE
        TST.B   D3
        BMI     STSP            IF IDLE START SPOOLING
        MOVE.B  D3,SPAF(A3)
        DC      WARMST

STSP    LEA     AQP(A3),A0
        MOVE.W  #9,D0           MOVE 10 PTRS UP
SS2     MOVE.L  4(A0),(A0)+
        DBRA    D0,SS2
        CLR.L   (A0)            CLEAR LAST PTR
        SUB.B   #1,NRINQ(A3)

* INITIALISE SPOOL FCB
        LEA     SPFCB(A3),A4
        MOVE.B  1(A2),FCBDRV(A4)
        MOVE.W  14(A2),FCBCTR(A4)
        MOVE.B  #255,COUNT(A3)
        MOVE.B  #0,SPAF(A3)     ACTIVATE SPOOLER
        DC      WARMST

FULL    LEA     QFULL(PC),A4
        DC      PSTRNG
        DC      WARMST

ERROR   DC      PERROR
        DC      WARMST

*-------------------------------------------------------------
* DATA AREA
*-------------------------------------------------------------
SPNAME  FCC     /PRTSPOOL/      spool device driver filename
NOPSPL  FCC     /SPOOL DEVICE PSPL NOT INSTALLED/,4
BADFIL  FCC     /Syntax:- PRINT filespec/,$D,$A
        FCC     /Inserts filespec (default .OUT) in spool print
queue/,4
QFULL   FCC     /PRINT SPOOL QUEUE FULL/,4
        END     START
```

If you want to avoid the effort of typing these listings in, the code for both of these programs is available for downloading from the Star-K BBS at (914) 241-3307.

A Modified LIST Utility

by David Underland

I have found the LIST utility from the SK*DOS users manual to be very useful. However, I did find a few things annoying. First, the printer does not formfeed after the listing is finished, and second, the lines per page count does not reset. Thus, the second listing will have the bottom and top of page margins inserted in the middle of the page. The following is a quick and dirty fix for this problem and also adds the ability to date stamp the top of a printout to help keep track of different versions of a program.

I have the following line in my STARTUP.BAT file;

```
dosparam 2 wd=80 pl=56 sl=10
```

This allows printing of 56 lines of text on a page with 10 lines skipped over the perforation. The problem is that the line counter in the print driver doesn't get reset after a listing is finished. To solve this situation, I added the following lines to the close subroutine at the end of the list program:

```
        MOVE.B    #$0C,D4
        DC        PUTCH             OUTPUT A FORM FEED
        MOVE.B    #$00,$D9F(A6)     SET PLINES FOR DEVICE 2 TO ZERO
```

The above outputs a form feed to the printer (the screen ignores it) and sets the line count (PLINES) to zero in readiness for the next printout. Note: if a print from another driver (assembler listing) is generated, it may NOT reset PLINES either. To be sure that the printer and the driver are ready for the next print out, do a manual formfeed on the printer and at the DOS prompt type:

```
LIST LIST
```

This will set the printer to top of form and reset PLINES in readiness for the next print out.

While I was at it, I decided that since I am very careless about version numbers and dates of file creation that I would also add a date stamp to the listing routine. The following is inserted immediately after the opening of the file to allow the file name and date that the file was last stored to be printed at the top of the listing:

```
*LETS PRINT THE FILE NAME
        MOVE.L    A0,A4             MAKE A4 POINT O FCB
        ADDA.L    #4,A4             FILE NAME STARTS AT BYTE 4
        MOVE.L    #7,D2             SET COUNT FOR 8 BYTE NAME
LOOP    MOVE.B    (A4)+,D4          GET THE CHARACTER OF NAME
        BEQ       EXT               IF ZERO, NAME SHORTER THAN 8 BYTES
        DC        PUTCH             PRINT IT
        DBRA      D2,LOOP           IF NOT DONE BRANCH
*NOW ADD EXTENSION
EXT     MOVE.B    #$2E,D4
        DC        PUTCH             PRINT A .
        MOVE.L    A0,A4             GET FCB POINTER BACK
        ADDA.L    #12,A4            EXTENSION STARTS AT 12 TH BYTE
        MOVE.L    #2,D2             SET UP FOR 3 CHARACTERS
LOOP1   MOVE.B    (A4)+,D4          GET THE CHARACTER
```

```
        BEQ       DATE           IF ZERO NO EXTENSION
        DC        PUTCH          PRINT IT
        DBRA      D2,LOOP1       BRANCH IF NOT DONE
*LETS PRINT THE DATE THAT THE FILE WAS LAST SAVED
DATE    LEA       MSG(PC),A4
        DC        PNSTRN         PRINT THE MESSAGE
        CLR.L     D4
        MOVE.L    A6,A4          MAKE A4 POINT TO FCB
        MOVE.B    FCBMON(A4),D4  GET MONTH OF FILE CREATION
        MOVE.L    #00,D5         PRINT WITH OUT LEADING ZEROS
        DC        OUT5D          OUTPUT IN DECIMAL
        MOVE.B    #$2F,D4
        DC        PUTCH          PRINT /
        MOVE.B    FCBDAY(A4),D4  GET DAY OF MONTH
        DC        OUT5D          PRINT DAY IN DECIMAL
        MOVE.B    #$2F,D4
        DC        PUTCH          PRINT ANOTHER /
        MOVE.B    FCBYER(A4),D4  GET YEAR
        DC        OUT5D          PRINT YEAR IN DECIMAL
        DC        PCRLF          PRINT CR AND LF
```

The above will print a line at the top of the listing such as:

```
LIST.TXT WAS LAST UPDATED ON      5/   28/    90
```

While this may not be fancy it does help keep track of different versions of a program.

The complete, modified code is then this:

```
                *LIST UTILITY FOR SK*DOS /68K
                *COPYRIGHT (C) 1986 BY PETER A. STARK
                *
                *EQUATES TO SK*DOS
                *
        LIB     SKEQUATE
*
LIST    BRA.S START
VER     DC.W  $0101       VERSION NUMBER
*
*START OF ACTUAL PROGRAM
START   DC      PCRLF
        CLR.B   D1
        MOVE.L A6,A0
        MOVE.L A6,A4
        DC      GETNAM
        BCC.S   NAMEOK
        MOVE.B #21,FCBERR(A4)
*
*ERROR ROUTINE
*
ERROR   DC      PERROR
        BSR     CLOSE
        DC      WARMST
*FILE SPEC WAS OK; DEFAULT TO .TXT
NAMEOK MOVE.B #1,D4
        DC      DEFEXT
*
*NOW ACTUALLY OPEN THE FILE
        DC      FOPENR
```

```
        BNE.S   ERROR
*LETS PRINT THE FILENAME
        MOVE.L  A0,A4
        ADDA.L  #4,A4
        MOVE.L  #7,D2
LOOP    MOVE.B  (A4)+,D4
        BEQ     EXT
        DC      PUTCH
        DBRA    D2,LOOP
EXT     MOVE.B  #$2E,D4
        DC      PUTCH
        MOVE.L  A0,A4
        ADDA.L  #12,A4
        MOVE.L  #2,D2
LOOP1   MOVE.B  (A4)+,D4
        BEQ     DATE
        DC      PUTCH
        DBRA    D2,LOOP1
*LETS PRINT THE DATE THAT THE FILE WAS CREATED.
DATE    LEA     MSG(PC),A4
        DC      PNSTRN
        CLR.L   D4
        MOVE.L  A0,A4
        MOVE.B  FCBMON(A4),D4
        MOVE.B  #$00,D5
        DC      OUT5D
        MOVE.B  #$2F,D4
        DC      PUTCH
        MOVE.B  FCBDAY(A4),D4
        DC      OUT5D
        MOVE.B  #$2F,D4
        DC      PUTCH
        MOVE.B  FCBYER(A4),D4
        DC      OUT5D
        DC      PCRLF
        DC      PCRLF
*
*MAIN LOOP TO READ AND PRINT EACH CHARACTER
MAIN    MOVE.L  A0,A4
        DC      FREAD
        BEQ.S   CHAROK
*
*IF THERE WAS AN ERROR, SEE IF END OF FILE
        CMP.B   #8,FCBERR(A4)
        BNE     ERROR
        BSR.S   CLOSE
        DC      WARMST
*
*CONTINUE IF CHARACTER IS OK
CHAROK  CMP.B   #$0A,D5
        BNE.S   PRNTIT
        MOVE.B  D1,D6
        MOVE.B  D5,D1
        CMP.B   #$0D,D6
        BEQ.S   MAIN
PRNTIT  MOVE.B  D5,D1
```

```
        MOVE.B D5,D4
        DC     PUTCH
        CMP.B  #$0D,D4
        BNE.S  MAIN
        MOVE.B #$0A,D4
        DC     PUTCH
        BRA.S  MAIN
*
*CLOSE SUBROUTINE
*
CLOSE   MOVE.L A0,A4
        DC     FCLOSE
        MOVE.B #$0C,D4
        DC     PUTCH
        MOVE.B #00,$D9F(A6)
        RTS
*
MSG     DC.B   " WAS LAST UPDATED ON "
        DC.B   4
        END    LIST
```

More Programming Tricks

by Gordon Reeder, 618 Adrian Drive, Rolla, MO 65401

Here are some more tricks from my programing bag. What I have here is a group of four routines that help a programmer use the free memory that remains in a system after a program is loaded. I was writing a utility that would search for data in several files. To do this I needed several buffers; one to hold the list of files, one to hold the data I was looking for, and one to hold the data that was found. At this point I was faced with "programers dilemma #27", how large should I make the buffers?? Should the file list buffer hold 8, 20, 50, 100 files? If it was too small the usefulness of the program would suffer, if it was too large I would be wasting space. Wouldn't it be nice if I could have the buffer size determined at run time!

It turns out that defining buffers at run time was easy to do. And now with these four routines, you can do it too! The buffers are placed in the free memory that is left over after SK*DOS loads your program. The amount of memory left depends on the size of your program and the amount of DRAM in your system. The last available byte of memory is at MEMEND(A6), and the first byte is at ... well now, just where does the free memory begin anyway? SK*DOS doesn't have a variable to point to the beginning of free memory, so we have to define our own, like this...

```
FREE    DC.L    FREE+4          end
```

As you can see FREE has to be THE VERY LAST variable declared. In fact it has to be the last thing in the program before the 'end' statement. When the program is loaded into memory by SK*DOS, the first byte of free memory will be right after the variable FREE, and FREE will be pointing to it. Now that we can find the free memory we need a way to use it. The first routine is called Get_Mem. It's almost trivial in its simplicity. All it does is read the value in FREE(pc) an return it in A4. So what? Well I included it for completeness, and

if you should want to add features to these routines you may find that you need a more complex Get_Mem. Placing it here makes upgrading easier. The next routine lets you set aside a block of memory. The size of the block, in bytes, is placed in A4, then Res_Mem is called. Res_Mem will check to make sure that there is enough memory to hold the block you want. Here is how to use it to set up a buffer.

```
    bsr     Get_Mem     you need to know where buffer will start
    lea.l   Buffer(pc),A1   this is where we will save it
    move.l  A4,(A1)         A4 has start address of the buffer
*                           it is now also in Buffer(pc)
    move    #1000,A4        lets make the buffer 1000 bytes long
    bsr     Res_Mem         buffer is now set
    beq     error           unless there wasn't enough memory
        .
        .
        .
Buffer  ds.l    1           of course you need to declare this.
```

Now if you should call Get_Mem it will return the new value that is the first byte after your new buffer. The next routine does the same thing that Res_Mem does, but in a different way. Instead of telling how much memory you want to set aside, you tell Set_Mem what the absolute end address is. The example routine above using Set_Mem would look like...

```
    bsr     Get_Mem     you need to know where the buffer will
start
    lea.l   Buffer(pc),A1   this is where we will save it
    move.l  A4,(A1)         A4 has the start address of the
buffer
*                           it is now also in Buffer(pc)
    add.l   #1000,A4        lets make the buffer 1000 bytes long
    bsr     Set_Mem         buffer is now set
    beq     error           unless there wasn't enough memory
        .
        .
        .
Buffer  ds.l    1           of course you need to declare this.
```

Actually, Set_Mem has a more useful function, As you will see shortly. The last routine doesn't even use the variable FREE, but it is useful for placing data into buffers so I included it here. Put_Mem can be used in a loop to place data into the free memory. D5 is used to hold the character to be placed into memory (compatible with GETNXT) and A4 holds the address to receive the data. When the routine returns A4 will have been incremented to the next address. Put_Mem also checks for memory overflows. Here is a way that you can use Put_Mem and Set_Mem to create a buffer that is exactly sized to its needs.

```
        bsr     Get_Mem     you need to know where buffer starts
        lea.l   Buffer(pc),A1   this is where we will save it
        move.l  A4,(A1)         A4 has start address of buffer
*                               it is now also in Buffer(pc)

Loop
* read a byte of data (from a file or the keyboard, whatever)
* the actual routine depends on your application
```

```
* be sure to preserve the contents of A4

        bsr.l   Put_Mem         put the data into free memory
        beq     error           oops, did we run out of memory?
        bra.s   Loop
*   After all the data has been read
AllDone bsr     Set_Mem         buffer is now set
        beq     error           unless there wasn't enough memory
        .
        .
        .
Buffer  ds.l    1           of course you need to declare this.
```

Now I don't know where you will be reading you data from, or how it's formatted. That's why I didn't include the code that gets the data or checks for the end of the input in the above routines.

Maybe a word about what's going on is in order. The call to Get_Mem returns the current value of the FREE. It is kept in A4 while one byte of input data is read. The input data routine needs to check for the out of data condition. If no more data is forthcoming then it should branch to AllDone. If not, it should place the data into D5 (GETCH and GETNXT will already have done this). The call to Put_Mem places the data into free memory. Notice that this is free memory, not memory that has been reserved or otherwise set aside. The call to Put_Mem will also increment A4 to the next byte. When all the data has been read A4 is pointing to what will become the new beginning of free memory. The call to Set_Mem uses that value to update FREE, and set the new start of free memory. You are not limited to just one buffer. After one buffer has been set up you can use these routines to set up a second, and a third... You get the idea. In the program that I was writing that started all this, I set up four buffers. The end of the last buffer is never declared, I just use all available memory. Put these routines into a file and name it USEMEM.TXT. Now when ever you write a program that needs to use the vast expanse of memory in your system just include it with the statement

```
lib     usemem.txt
```

The routines will be available to you with out having to type them into your source code. Well that's all for this time. But my bag of tricks is far from empty. So when I find some more useful routines I'll pass them along. What about you out there, yes, you reading this news letter. Do you have any tricks you know? How about passing them along too.

Free memory routines:

Get_Mem	returns the first available byte of free memory
Res_Mem	Reserves a block of x bytes of memory
Put_Mem	puts a byte into free memory
Set_Mem	sets the start of free memory at a user defined point

All these routines use a longword 'FREE' that must be defined in the main body of the program. Like this...

```
*FREE   DC.L    FREE+4      init to point just past program
*       end     start       end of program
```

As you can see, the longword FREE must be the last declared variable * in the program. It comes right before the 'END' statement. * After SK*DOS loads the program, the first free byte of memory will be * directly after the variable FREE. And FREE will be pointing to it!

```
*    FIND FIRST FREE BYTE
*    To call.....
*         no arguments needed
*    Exit.....
*       A4 = first available byte of free memory
---------------------------------------
Get_Mem
     move.l  FREE(pc),A4      get value from FREE...
     rts                      .... and return
*                      trivial, I know, but by placing it in a
*                      subroutine I can add features later.

*********************************************
*
*    Reserve memory
*    To call...
*         A4 = how many bytes to reserve
*    Exit...
*         A4 = First free byte past reserved block
*         Z flag =0 = Memory overflow
---------------------------------------
Res_Mem
         move.l  A1,-(A7)     save A1
         move.l  FREE(pc),A1 get Free pointer
         add.l   A1,A4        add offset
         cmp.l   MEMEND(A6),A4   see if there is enough memory
         bge.s   Mfull
         lea.l   FREE(pc),A1      point too FREE pointer
         move.l  A4,(A1)          save the new value
         move.l  (A7)+,A1         restore A1
         rts
*    What to do if we run out of memory
Mfull    andi    #11111101,CCR    clear Z bit in CC reg.
         rts

*********************************************
*
*    SET MEMORY ROUTINE
*
*    to call...
*         A4 = end of block +1 of a block of memory to reserve
*    Exit...
*         Z flag =0 = Memory overflow
*         nothing changed
*---------------------------------------
Set_Mem
         move.l  A1,-(A7)     save A1
         cmp.l   MEMEND(A6),A4   see if there is enough memory
         bge.s   Mfull
         lea.l   FREE(pc),A1      point too FREE pointer
         move.l  A4,(A1)          save the new value
```

```
        move.l  (A7)+,A1        restore A1
        rts
*   What to do if we run out of memory
Mfull   andi    #11111101,CCR   clear Z bit in CC reg.
        rts
*********************************************************
*
*   MEMORY INPUT ROUTINE
*   To call.....
*       A4 = Buffer pointer (auto inc)
*       D5 = char to place in buffer
*    Exit...
*       A4 = next Byte of memory
*       Z flag =0 = Memory overflow
*   Note:
*This routine doesn't use the variable FREE. But A4 can be set
*with a call to Get_Mem. After all the data has been placed into
*memory, a call to Set_Mem will protect it.
*---------------------------------------
Put_Mem cmp.l   MEMEND(A6),A4   see if there is room in memory
        bge.s   Mfull
        move.b  D5,(A4)+        place char in memory
        rts                     return

*   What to do if we run out of memory
Mfull   andi    #11111101,CCR   clear Z bit in CC reg.
        rts
```

SK*DOS Update

by Peter A. Stark

The following change to SK*DOS will only be of interest to advanced programmers, and I am including it here only in the interest of completeness.

We have recently updated SK*DOS to version 3.2 by the addition of two new variables called DATBEG and MEMBEG. Both of these were put in at the request of Dr. Jack Crenshaw, who is writing a linker/loader called JINK for use with SK*DOS.

Both variables can be accessed from programs by using the A6 register (which normally returns to user programs, pointing to the user data area of SK*DOS) as follows:

DATBEG is 5100(A6); in most versions of SK*DOS, this translates to $27EC.

MEMBEG is 5104(A6); in most versions of SK*DOS, this translates to $27F0.

The names DATBEG and MEMBEG are both being added to the SKEQUATE.TXT file.

Both of these variables are generally uninitialized, and are ignored by SK*DOS except when loading a binary file from a disk into memory. The SK*DOS binary file format now has two new load codes, which specifically refer to these two locations:

The load code $30 takes the next long word in the binary file, adds to it the current value of OFFSET, and puts the resulting 32-bit sum into DATBEG.

The load code $31 does exactly the same thing, but puts its sum into MEMBEG.

In both cases, SK*DOS does not do any checking on the results. That is, it does not check for overflow, for an even or odd number, or even that the result is greater than OFFSET or smaller than MEMEND.

Jack expects to use these two new variables to keep track of the memory available to load modules.

The Star-K BBS System

Some of you may not be aware that you can talk to other SK*DOS users via the Star-K BBS. There are also files to download, such as user-contributed files, or copies of all previous issues of *68 News*.

The telephone number is (914) 241-3307, and the protocol is 300, 1200, or 2400 baud, 8 bits, no parity. You may access the BBS with your SK*DOS system, using a communications program such as CMODEM or EZMODEM; via a modem connected to a terminal, or even (heaven forbid!) with a PC or clone.

I hate to admit it, but the BBS itself runs on an XT clone computer, running at 4.77 MHz with an 8088 processor. The system has a no-name 2400-baud modem, and a 10-megabyte hard drive.

In a sense, the hard drive is a tribute to the efficiency of 68000 programming. 10 megabytes is plenty of room for our programs, whereas 10 megabytes would be woefully inadequate if we were supporting IBM PC programs. In this case, small *is* beautiful!

The *68 NEWS* is published and copyright © 1990 by Star-K Software Systems Corp, P. O. Box 209, Mt. Kisco NY, 10549. The editor is Peter A. Stark.

The subscription price is $10 per year. Your subscription runs through the issue number printed above your name on the label. For example, N15 means your subscription runs through issue 15.

We accept display advertising at the rate of $10 per half-page (3.5" high by 4.5" wide). Readers are invited to contribute articles, letters, programs, tutorials, and other material for publication. We publish only material and advertising which, in the opinion of the editor, (a) applies to hardware or software for 68xx(x) type processors, and (b) is of a nature which would not normally be of interest to the major computer magazines. We simply do not have room for items of a very general nature, or items which pertain to very popular systems like the Macintosh or Amiga.

Please send articles or other material to us at the above address (preferably on disk); you may also fax us at (914) 241-8607, send it via modem to our BBS at (914) 241-3307, or call us at (914) 241-0287.

We thank you for your support.

From: Star-K Software Systems Corp.
P. O. Box 209
Mt. Kisco, NY 10549

Address Correction Requested

To: