The following is a description of the programs on Proteus cassette C7. These programs have been donated by members of Proteus who wanted to share the fruits of their labor. Proteus has attempted to check some of the programs for validity, but you should exercise caution using any of these programs since we can never be sure they are error free. If you discover errors, please let us know so we can try 0 keep the programs updated.

**********

SCS16 -- SELF-CONTAINED SYSTEM, VERSION 1.6

Description

This is an assembler and disassembler for Sol, based upon Software Package #1 that was long ago distributed into the public domain by Processor Technology Corporation. It contains an editor that creates files of ASCII text in RAM, an assembler that assembles from these source files in RAM into object files elsewhere in RAN, and a disassembler that recreates source files in PAM from object programs in RAPS. The disassembler is based upon the one by Ward Christensen published in Feb 1977 issue of Dr. Dobbs Journal. The assembler is extremely fast since it doesn't have to do any input/output except to the listing device, which is usually video. This makes it very handy for rapid debugging and alteration of source programs, provided the source code will fit into available memory space.

Loading

SCS16 loads into the bottom of memory beginning at 100H. The editor and assembler load into 0100H-1589H. The disassembler loads into 2000H-27FFH. Files can be created at 2800H and beyond. SCS can be loaded as a CP/M command file, since it doesn't disturb the bottom page of memory (0000-0100). The area between the assembler and dissembler is used for tables (symbol table, buffer, stack, etc.). A maximum of 255 symbols can be created. I/O is done through SOLOS. The entry point for initial entry to the SCS command interpreter is 0100H (clears files). If you leave control of SCS and want to re-enter without losing the files in RAM, enter at 0103H. The disassembler may he overwritten by files if you don't want to use it. Entry to the disassembler will be described below.

Commands

On initial entry, the system will display the prompt, a right-brace. At this point, SCS is executing the input command processor and awaiting user input of commands. All commands are four characters, however, some commands have a fifth character option which invokes a variation in the command.

During command entry, the DEL key will backspace the cursor and erase the last character from the screen. The RETURN key signals the end of the command and causes the command to be executed.

In the following command descriptions, the literal command name will be in upper case and the arguments which you must

supply will be named in lower case. Arguments are always taken as hex numbers (addresses) or names.

FILE /name/ org

Creates a new file in RAM named "name" and located beginning at address "org". You can have up to 6 files, one of which is the "current" file. File names can he up to 5 characters long. After this command is executed, the new file will be in the internal directory with its beginning and ending pointers both set equal to "org", and it will be marked as the Current file. You are responsible for managing the files so that they do not grow to overlap. Select an origin far enough away from the end of files below it.

FILE /name/

Makes the named file the Current one. This is equivalent to re-opening the file and closing the previous one. SCS will display its beginning and ending addresses.

FILE /name/ 0

Origin zero signals SCS to delete the file.

FILES

Prints all files and their boundary addresses. The first file listed is they Current file.

AUTO from incr

Begins automatic line numbering to create new lines in the Current file. The first line number will be "from" and each successive line number will be incremented by "incr". SCS will display the line number and wait for you to enter a space. Everything following the space until you enter a RETURN will be placed into that line in the file. If the line number already existed in the file, your new text will replace the entire line that was there. If the line number is a new one to the file, your text will be entered into the appropriate numerical location in the file, along with the line number. Typing any character other than a space after the line number will exit from the automatic line numbering mode and the line number will be erased. You can temporarily break sequence to enter a different line number, say to alter a previous line, by entering the desired line number instead of the space. After the text for the line is entered, the RETURN will cause the text to be placed in the file and the auto-numbering will resume where it left off.

LIST line

Lists the current file beginning at line number given. If no line number is given, the whole file will be listed. In this version there is one way to pause the listing: turn

sense switch #8 off. Whenever switch #8 on the Sol is off, the LIST command will only list one line. In a future version in preparation, the listing will pause when a space bar is pressed, but that doesn't work in this version. The Solos display speed can be set to slow down the listing and the Solos system output pseudoport can be set to direct the output to a printer.

LTXT line

Lists the text in the file, without line numbers, beginning with line number given. If line number not given, then the whole file will be listed without line numbers.

SOLS

Returns control to Solos. You can get back to SCS16 by giving the Solos command: EX 103.

SFIL

Saves the Current file onto tape #1 with type=F and the file's own name. Outputs to the tape with Solos block mode format.

GFIL

Gets the next file from unit #1 and puts it into the load address from the tape header. It replaces the Current file in the directory, so you should create a new file entry prior to giving the GFIL command, unless you want the Current file to be lost.

EOCT addr

Enters octal data into memory starting at "addr" address. It works line the ENter command of Solos, but takes octal input rather than hex.

DOCT from to

Dumps memory in octal. The "from" and "to" addresses define the area of RAM to he dumped. These addresses are given in hex as usual.

DASC from to

Like DOCT but dumps memory in ASCII.

DELT from to

Deletes lines beginning with the line numbered "from" through the line numbered "to".

Giving a line number followed by text ending with a RETURN will cause the line to be entered in the file in its appropriate numerical sequence. If line number already was in the file, the old line will be replaced by the new, line and the file space will he compacted if necessary to avoid wasted space.

Line numbers can range from 0000 to 9999 (decimal) and preceding zeros must be given to make the line number 4 digits. It is suggested that lines originally be numbered with five or 10 units difference so that there are numbers available to insert new lines between any two lines.

ASSM from to

Assembles a file (assumed to contain an assembly language program). The source file begins at address "from", and the resulting object code is placed in memory beginning at location "to". Assembly terminates when the "END" pseudo-operation is reached in the source file, or when the end of the file is reached. Program should begin with an "ORG" pseudo-operation to create object code that can reside at the desired location.

Instructions in the source file have the following syntax:

line_label_opcode_operands_;commentsRETURN

where

line represents the line number given by the editor,
_ represents the space that follows the line number,
label represents the optional symbolic name of this location in the memory, up to 5 characters, the first must be alphabetic and no special symbols allowed,
_ represents one or more blank spaces or tabs (if tabs are used they are expanded in the listing); if no label field, you must still give these spaces,
opcode represents the INTEL 8080 mnemonic operation code or pseudo-operation code,
_ represents one or more blank spaces or tabs,
operands represents the parameters needed by the instruction, separated by commas if more than one,
_ represents one or more blank spaces or tabs,
;comments represents the programmer's remarks which will be listed but otherwise ignored, semi-colon optional.

A colon can be placed after the label for compatibility with the Intel assembler. A semi-colon or asterisk in the first column (after the required blank following line number) will cause the entire line to be taken as a comment line, that is, it will be listed but otherwise ignored.

Some symbols are pre-defined by the assembler and may not be used by the programmer except in the operand field. These symbols are: A,B,C,D,E,H,L,M,SP,PSW. They represent the registers of the 8080, the memory addressed through HL, the stack pointer, and the program status word (flags and A). In addition, the dollar-sign ($) is defined to be the current value of the program counter after the current instruction is assembled; that is, it points to the first byte of the

next instruction.

Operands can be constants, symbolic names, or expressions. Constants can be decimal integers (signed or unsigned), octal numbers, hexidecimal numbers (signed or unsigned), or ASCII constants. Hexidecimal numbers must have the letter "H" as suffix and must not begin with a letter; a preceding zero may be used if number would otherwise begin with a letter. For example, "0C000H" is the starting address of Solos. Octal numbers must be suffixed with the letter "O" or "Q". Decimal numbers may be suffixed with the letter "D", but that is optional since numbers without a suffix are taken to be decimal. ASCII constants are enclosed within single quote marks; for example, 'C'. Expressions are symbols and constants separated by arithmetic operators "+" or "-" and are computed using 16 bit arithmetic modulo 65536.

Pseudo-operations are instructions to the assembler rather than mnemonics for 8080 instructions. This assembler recognizes the following pseudo-ops:

ORG--"Origin". Sets the assembler's program location counter to the value of the operand. The label if given will be equated to the origin.

EQU--"Equals". Defines the label symbol to have the value of the operand expression.

END--"End of assembly". Signals the end of the source file.

DS--"Define Storage". Advances the assembler's program location counter by adding the value of the operand, effectively reserving a given number of bytes of memory without placing any data in them.

DB--"Define Bytes". Defines bytes of data to be placed in memory beginning at the assembler's current program counter location. For example,

```
DB 'ABCD'
```

defines four bytes each containing the ASCII letters 'A', 'B', 'C', 'D', respectively. Mixed constant types may be given if separated by commas, as

```
DB 'message',0DH,18
```

will define ASCII constants, followed by one byte hex constant, and one byte having decimal 18.

DW--Define Word. Similar to DB but defines two bytes of storage only, and the bytes will be placed in memory in reverse order, which is the standard Intel 8080 placement for two-byte addresses stored from registers or to be loaded into 2-byte registers.

The assembler listing will flag errors detected by the assembler. The flags are:

O Opcode error
L Label error
D Duplicate label
M Missing label
V Value error
U Undefined symbol
S Syntax error

R Register error
A Argument error

ASSME from to

Acts the same as the ASSM command above, except that the listing will only show lines having error flags.

EXEC addr

Instructs SCS to begin execution of an object program at address "addr". A return-from-subroutine (RET) instruction will return control to SCS if the stack is properly maintained in the executing program.

EXEC 2000 from -- Executes the disassembler.

The disassembler is more fully documented in the Dr. Dobbs Journal article mentioned above, and it is essential that you read that article to use it. To execute the disassembler, give this command, where "from" is the hex address of the object program to be disassembled. Sense switches are used to control the phases of the disassembly. Control-Z character will pause the listing and space bar will resume it. Control-X returns control to the SCS command processor. When reading the article, keep in mind that the sense switches are numbered 1-8 on the Sol PC board, whereas they are referred to as 0-7 in the article. Also, when a Sol switch is OFF it produces a 1 or HI value when read. This corresponds to the UP position on the Altair or Imsai computer's sense switches.

ALS-8 Compatibility

SCS stores text files in memory the same way that Processor Technology's ALS-8 program does. This makes it possible to read source files written by ALS-8. To do this, the tape should be positioned so that the desired file is ready to be read on the cassette recorder. Use the CAtalog and GEt commands of Solos to position the tape. Then create a file in SCS to receive the program and give the GFIL command. SCS will read the file into memory and set the directory entry to contain the file's name from the tape header and the file's address limits in memory. You can then manipulate the file with SCS.

IMPORTANT: If the file was written having an address that overlaps SCS or its tables, you must relocate the file first by reading it into memory with Solos at an address that will not harm anything in memory (above your last file) and writing it back out onto tape with the desired location in the header. Read the Solos manual for details, but in brief, the following Solos commands will do this:

GET file loc
SAVE file loc to addr

where "file" is the filename on tape,

"loc" is the hex address where you can store the file temporarily,
"to" is the last address occupied by the file after Solos has loaded it (add the file length given by Solos on completion of the GET, to the starting address "loc", minus one, using hex arithmetic),
"addr" is the address where you want SCS to place the file when it reads it from tape (header address).

**********

CSEL--CORRESPONDENCE SELECTRIC OUTPUT DRIVER

This is a custom output driver for Solos which will operate a Correspondence-code, IBM 2741-compatible selectric terminal properly hooked up to the Sol serial port. Examples of such terminals include the Carterfone S15C terminal manufactured by Datel and distributed by Carterfone, the Anderson-Jacobson 841-C (not the ASCII modified one, the original one), and similar terminals. This driver will not work correctly with the EBCDIC-code terminals, nor with the ones modified to accept ASCII-code.

The proper electrical hook-up and Sol switch settings were described on page 3 of Solus News, vol. 0, no. 2, Oct/Nov 1977. The Terminal requires baud rate 134.5, word length 6 bits plus odd parity plus one stop bit, standard RS-232 levels. To achieve the non-standard baud rate of 134.5, the Sol's 110 baud rate must be modified with a simple, reversible hardware alteration: pin 12 of U84 (4029) is bent out so it doesn't engage the socket and it is wired to +5 volt supply trace nearby. The wire can be applied to the IC's pin using a simple wire-wrap tool, so the pin is not harmed, and the other end of the jumper wire soldered to the +5 volt feed-through hole. See the article cited for more details.

To use the driver, load it with Solos and EXecute C863. This will initialize the driver and issue a message on the video, "TURN ON SELECTRIC TERMINAL". The terminal should be off when you execute this initialization, and you should turn it on as instructed. When the terminal is powered on, it sends a special character to the Sol, which the driver expects to find. When the character is received, the driver sends another message to the video, "STRIKE ITS RETURN KEY". At this point the Proceed light (or equivalent) on the terminal should be on. Press the Carriage Return key on the typewriter. The Proceed light should go out, the carriage should return to the left. The driver will send a special character back to the terminal (the digit "9") which does not print, and then the typewriter should type "READY!".

The driver was written for a specific typesphere (print element) which has the exclamation mark where ordinary office typespheres have the one-half symbol. To see how your typesphere matches the one assumed by the program, you can type out the entire character set in order and see what comes out. I'm sorry I don't know for sure which typesphere is the correct

one, but I think it is #072. (Another Selectric driver to be donated to the Proteus library will have the source code and instructions for altering the character set.) Nevertheless, the driver will work correctly with any ordinary office selectric typesphere for all of the letters, numbers, and most of the commonly used punctuation. ASCII characters not available on the assumed typesphere will print as blanks so that you can fill in the proper character by hand. (The other driver to be donated will create representations of all of the ASCII characters using overstrikes if necessary.)

After the "READY!" message appears, the initialization routine sets Solos so that the Custom Output driver address is C903 and the output pseudo-port is 3, and then it returns to Solos for further command processing. You may then run programs which direct output to the system output port and the output will appear on the typewriter. For example, doing a CAtalog command will list the file information on the terminal rather thin on the video.

Notice that the location of the driver is in Sol scratchpad RAM and that the initialization part of it is in the area which is erased by a system reset when Solos initializes itself. If you must do a reset and restart a program, you can do the initialization yourself without reloading the driver. You must

```
>SET CO C903
>SET O=3
```

and restart your program. If the terminal is turned off for some reason, remember that it must receive a "9" character as the first thing it gets after power is turned on again; otherwise it won't print anything. The initialization routine does this for you, but if you do the initialization yourself you must arrange to get the "9" out there before any other characters. A nine on the typewriter keyboard won't do, it must be transmitted from the computer.

**********

CUP--CASSETTE UTILITY PACKAGE by L. Morgenstern, 304 Rheem Blvd
Moraga, California 94556.

CUP is represented by 6 files on tape C7: CUP0, CUP1, CUP2, CUP3, SCDO1, SCDO2. The complete documentation is in CUP0, as an ALS-8 text file. The files contain as follows:

CUP0 is in ALS-8 format and contains an introduction to CUP and SCDO.

CUP1 is in ALS-8 format and contains CUP documentation, including a brief documentation of SCDO, a powerful feature of CUP by which Solos commands can be executed from the screen, similar to FORTRAN DO loops.

CUP2 is in ALS-8 format and contains the assembly source code for CUP, including SCDO.

CUP3 is the assembled object file of CUP in Solos single-block format, the way programs are usually saved.

SCDO1 is in ALS-8 format and contains SCDO documentation.

SCDO2 is in ALS-8 format and contains SCDO source file.

**********

LIST--STORES A DIRECTORY OF FILES AT THE BEGINNING OF A TAPE

LIST makes a file at the start of your cassette so that you know where your programs are located by counter reading on the cassette recorder.

Record a cassette, leaving some space at the beginning for the file to be created, and noting the counter reading for the start of each program. Load and execute the LIST program. Rewind the tape to be LIST'ed, and set it up to record at the beginning of tape. slake entries in the form shown by LIST. When the last entry has been made and entered by carriage return, press ESCAPE and the program will generate a file called "CFILE", which will load and execute at C900. Once CFILE has been executed, as long as the system has not been reset (by UPPER CASE & REPEAT keys) the listing may be reviewed by giving the Solos command "CF".

Correction to LIST program

Examine the byte located at LIST's origin + 1B2H. If it reads "30" change it to "36". (Any CFILE's made with the old version should have C933 changed from "30" to "36" also.) The custom command address in the old version was incorrectly set to C900 so that it CFILE is recalled, it re-enters the custom command routine in Solos which, as per specifications, removes the command. This correction will prevent that.

**********

BSHIP--BATTLESHIP GAME

Battleship is a classical game wherein each player has a playing board that cannot be seen by the other player. Each player tries to shoot the opponents battleships without knowing where they are, and the opponent is required to designate which shots hit and which missed. Shots are given by the coordinates of the target. The first player to destroy the opponents fleet is the winner.

In this version, you play against the computer. Execute the program and you can ask for the rules of play.

**********

TAPE2--GENERATES A TEST PATTERN ON CASSETTE TAPE

This program writes a pattern on the cassette and then reads it back again to see if the recorder has been accurate.

Operation:

1. Use the GET command to load "TAPE2".
2. Set up the cassette recorder #1 in the record mode. (The program does not use the motor-control relays. Leave the motor cable unplugged from your recorder.)
3. Let it record for 15 seconds or so.

4. EXecute C900. (The program will run forever until you stop it.)
5. When you have recorded to the end of the cassette, turn off the recorder and do a system reset at the keyboard (UPPER CASE/ REPEAT).
6. Rewind the cassette to the beginning.
7. Start the recorder in playback mode.
8. EXecute C930 when the recorder has passed the blank leader and is reading the empty 15 seconds your first recorded.
9. The resulting display will start incrementing each character position (starting at CF00) through all 256 possible character combinations and then increment to the next character position. This will continue for 4 lines and then repeat. Any errors that are detected are displayed as follows:

F: Framing or overrun error
D: Data error
C: CRC error
S: Sync error.

When an error is printed out, the next character that is printed is the byte that caused the error.

The tape format is:

4 bytes of 00
1 byte of 01
256 bytes 00
1 byte CRC
.
. repeats ad infinitum
.

**********

PIRAN--PIRANHA VIDEO ARCADE GAME

The instructions are contained in a single video screen image recorded in the file called "PRNIN". To view that file, do exactly this:
1. Prepare the cassette #1 to playback, motor control and audio cables properly in place.
2. Position the tape just before the PRNIN file, using the CA command.
3. Press the CLEAR key, and then press MODE SELECT key 15 times. This will give a trail of greater-than prompts down the left side of the screen to the bottom.
4. Type "GET PRNIN" and RETURN. 5. The file will be loaded into the video display area.

**********

SSnA--SINGLE-LINE SIMULATOR

This program simulates the execution of a machine language program residing somewhere in RAM. The original concept was based on an article in the Sept 77 issue of Kilobaud. It was adapted by John Zimmerman.

The simulator is recorded on tape C7 in files SS2A through SS6A. The only difference between these versions is the

location where the program will be loaded. SS2A loads at 2A00, SS3A loads at 3A00, and so on. Each one should be Executed at its first address to begin execution.

To start

1. Execute starting address (SA) of the simulator.
2. Program will expect exactly 4 hex digits for the starting address of the program to be simulated. The input routine is primative--there is no delete key. Just restart the execution if you make an error.
3. The simulator will simulate execution of the program one instruction at a time each time the space bar is pressed.

Functions

1. Space bar or most other keys will advance the simulator.
2. S = output Stack data
3. L = Back up one instruction (won't back more than one).
4. M = output memory locations in hex. You enter 4 hex digits for address to be examined
5. A = output 48 ASCII characters from memory. You give 4 digit address.
6. G = go the specified number (hex) of instructions before printing out another summary. Enter 4 hex digits. Each time you press the space bar simulator will advance that many instructions. You can give another command or another G command to do something else.
7. B = breakpoint. Give the address to insert breakpoint as 4 hex digits. Space bar continues.
8. R = range. Enter 4 hex digit low address and 4 hex digit high address (1 beyond). Simulator will run outside this range, but will only print summaries within the range.

Keyboard entry

A message is printed to indicate that the program being simulated wants a character from the keyboard. It is echoed after you give it, and simulation continues. A DEL key will cause no-character to be returned as the entry. This is customized for Solos: C02E=entry to keyboard routine, C036=exit address to keyboard routine. For other systems, alter location +04D0 relative to the origin of the program.

Exit and Restarting

The "E" key ends the simulation and returns control to Solos via C004. The current step's information is saved so that upon entry to the starting address of the simulator, pressing the space bar 4 times in place of the starting address will cause the simulator to resume where it left off.

**********

BAUD--ASCII-TO-BAUDOT OUTPUT ROUTINE

See Proteus News, vol 2, no 2, p 11.

**********

UTIL, MTEST--OBJECT FILES
UTSYM, MTSYM--SOURCE FILES

See Proteus News, vol 2, no 2, p 19 and p 14, respectively.

**********

ASSM, TASSM--ASSEMBLER AND TAPE ASSEMBLER

See accompanying reprint from PRINT OUT, the newsletter of the Central Texas accompanying Computer Association. Both are similar to SCS16 on this tape, but TASSM assembles from source files recorded on tape, thus allowing very large source programs to be assembled, larger than could fit into available memory space.

**********

# 8080 ASSEMBLER MODS

By Ron Parsons

I have made a number of modifications to a version of the editor/ assembler usually known as Processor Technology's Software Package #1. The source changes as well as the new command syntax and command operation are discussed. They are presented here to stimulate your own ideas for other possible changes.

Modifying software to increase its capabilities or make it easier to use can be very challenging. Learning to understand the original software in order to make the changes will improve your skill in programming. If you have made changes to existing software, write them up for PRINT-OUT. Share your ideas!

I will use the standard notation, that is, optional parts of syntax are enclosed in [ ]. The PAGE command, which moves a 256 byte segment from one location in memory to another, has been changed to MOVE and an optional parameter added to specify the number of bytes to be moved. The new syntax is

MOVE [/number/] (fm. addr) (to addr)

If (number) is not specified, the default is 256 (it reverts to the old PAGE command action). The value of (number) can range from 1 to FFFF (hex).

The FILE command has been augmented to provide for recovering an existing file in memory after its directory has been lost or trashed. (The directory is cleared when you restart the monitor at its beginning address, F000.) The directory contains three essential pieces of information - the beginning of file address (BOFA), the end of, file address, and the highest line number. The command

FILEO_/file name/(BOFA)

(mnemonic FILEOld) starts scanning the file at the beginning of file address, restores the directory, and makes (file name) the current file. This is a lot easier than reconstructing the directory by hard (and more accurate, too). The _ between the FILE command and the / MUST be a "blank."

A file renumbering command has been created to renumber a file. The command

FILER [1st line # [line # increment]]

resequences the current file's line numbers starting with the decimal first line number, and incrementing by line number increment. The defaults are both 10.

The LIST command has been changed to permit listing the entire file, a single line, or a group of lines without resorting to sense switch flipping. The syntax is

LIST [1st line # [last line #]]

If last line number is specified, lines from "first" to "last," inclusive, are printed. If last line number is ommitted, only the "first" line is printed. If the command is given without parameters, all lines are printed. For user convenience when using a video display, output from LIST or ASSM is suspended when any keyboard key is depressed. Printing resumes when a key is again depressed.

To help a non-typist enter text into a file; say, for subsequent assembling, the editor has been modified to provide the line number automatically when lines are being added at the end of the file. The automatic line number increment is 10. A blank is also provided after the line number. A carriage return entered as the first character of the line exits the automatic line number sequence. Tabs have been added at 10, 15 and 20.

A new command has been created to print the symbol table of the most recent assembly. Four symbols and their associated values or addresses are printed par line. Keyboard input suspends the output pending another keypress. The symbols are sorted.
The syntax is

SYMB

All of the above changes require an additional 200 (hex) bytes of code. Since memory is my scarcest resource, the assembler was modified to read source code from cassette rather than from memory. The tape is read into a memory buffer until an end--of-line character is read. The line is "quickly" processed and the tape is read again without missing a character. Standard Kansas City speed (300 bits per second) gives sufficient time to process (and video display with a Processor Tech VDM) each line. Higher tape speed (1200 bits per second) is too fast.

When the end-of-file is read from the tape, the assembler displays YES? and waits for keyboard input. If more files remain, you position the tape at the next file-and input "C" from the keyboard. When all files have been processed, you reposition the tape at the first file and "2" is input to begin the second pass. All files must be read again in the sane order.