Software

The low level monitor is located in the ROM. The monitor is the Amsat monitor published in September 1976 Byte magazine. This is an excellent monitor, and actually is an earlier version of a many times modified monitor by the Amsat people. I understand that a greatly expanded version is now being used, and

perhaps they will finalize it and republish it. The version I use is the one from the magazine, and find it quite adequate. There were some personal modifications made, but these are not too significant.

The high level monitor resides in RAM. The operating system makes use of memory instead of a disk as is normally found in conventional systems. There are some adequate disk operating systems around, but these are not suitable for memory oriented systems. The hobby disk is here, and more and more hobbyists will be purchasing floppy disk systems, but for the rest of us (still the majority) there are no good memory based operating systems around. To fill this gap, I wrote my own which I believe to be flexible and powerful enough to be used in any system with more than a minimal amount of memory. Depending upon how much of the operating system is desired at any time, up to 6K will be required. To be useful, a text editor is almost essential. The one I use was published in the June/July 1976 issue of Dr. Dobbs Journal of Computer Calisthenics & Orthodontia. I have made some modifications to it, but all that is really required to interface to it is to change it into a subroutine. In addition to a text editor, an assembler and BASIC are useful. I would say that the minimum of memory for which the operating system could really be useful would be 16K. The more memory, the greater the power of the operating system can be exercised.

I will describe what the operating system does. When in the monitor, a percent sign is given as a prompt. The monitor then waits for a command. All commands are simply an executable file name. Files may be called either in upper or lower case. However, they are always stored in the directory as lower case. Anything that is typed in to the monitor the monitor expects to find as an executable program in the directory. The directory is located in memory, and contains the file name, the byte count, the starting address, and the executable address for every file. The file name may be anything - there are no restrictions on the length of the file name, and all characters with the exception of the less than, greater than, number sign, space, and exclamation point can be used. These characters have special meaning to the monitor, and will be explained. The byte count is

the number of bytes of the file. This may or may not be a necessary entry in the directory depending upon which program is using the file, and if its executable location is different from its starting address. The starting address is where the file is found. The executable address is where an executable program is supposed to be located in order to operate properly. Note that the starting address and the executable address may be different. This allows many programs to exist in memory at the same time when they all must be loaded in the same place in order to execute. When a command is given, (that is, a file name is entered) one of four things will occur. If the file is not found in the directory, a message will be given to that effect. If the file is found and does not have an executable address (an address of FFFF) then a message is given. If the file is found and the executable address is equal to the starting address, then that program is already loaded in its proper place, and the program is called as a subroutine. When a return from the subroutine is found, the monitor resumes control. In this case, the byte count need not be accurate. The last possibility is that the file is found in the directory and the executable address does not equal the starting address. Here the operating system copies the file to its executable location by the information in the byte count, and the byte count must be correct. Then the copy of the program in the correct executable spot is called as a subroutine. After return to the monitor, the executable copy is dead material and is of no consequence. More than a simple call to an executable program can be done from the monitor. Parameters can be passed from the program call to the called program. A space simply separates the filename from parameters.

As an example, one of the files in the directory is the directory command. This command lists the entire directory or individual entries in the directory. When 'dir' is entered, the file named 'dir' is located in the directory, and it is called as a subroutine. That directory program lists the current directory. If 'dir xyz' is entered to the monitor, then only the directory listing of the file named 'xyz' will be printed. Any number of directory entries may be printed in this manner, as 'dir xyz garbage temp' will list the directory contents for

'xyz', 'garbage', and 'temp' if they exist, and an appropriate message if they do not. The syntax is identical for all of the other commands.

There are universal options available for any program call. The standard output is the terminal, or more precisely, whatever has been commanded by the 'stty' (set teletype) command. This is normally the console terminal. Any output which the called program creates is directed to that standard output. Likewise, any input which the called program requires is taken from the current standard input, normally the console keyboard. Other input or output besides or instead of the standard input/output may be specified for the duration of any individual command (program) by the following:

The > option. By typing 'file1 > file2' the program file1 is called and executed. For the duration of that file1 program, all output is diverted from the standard output to the file 'file2'. Nothing is printed until the program 'file1' returns to the monitor, when the standard input/output options are reset. For instance, the command 'dir xyz > dirout' will call the directory command, which will list the directory contents of the file 'xyz'. However, instead of this being printed on the terminal, it is placed in an ASCII file named 'dirout' which must have already been in existance. That file could then be inspected with one of the other commands, or even with the text editor. Note that anything which was previously written in the file 'dirout' is overwritten and lost. Another option >> will direct output to the named output file, but will add to that file rather than overwriting it.

Along with the mentioned output diversion options, the equal sign may be appended to the greater or greater greater output option. This will do the same as before, but the output will be printed on the terminal at the same time as it is being written into the named output file.

Similar to the output diversion, an input diversion may also be made. Normally all input comes from the keyboard, but if the < option is used, input will come from the named file instead. Input will be restored to the keyboard when the called program returns to the monitor. Both input and output diversions

may be made simultaneously. For instance 'basic < bas.inp> bas.out' will call the program basic, take commands from the file 'bas.inp', and place all output in the file 'bas.out'. There is another output option which simply eliminates all output. This is the exclamation point. If this is found after any command, no output will be produced during that program execution. This is interesting to use to test program execution speed, as no time is lost in printing material. It is also useful for certain commands which manipulate files and may also produce output which is not desired at a given time.

A nonexecutable file may also be specified for simplicity as '#wxyz', where 'wxyz' is a hexadecimal address. The monitor uses spaces for delimiters as previously mentioned, and any number of spaces may be used to separate files and parameters.

The operating system is essentially two parts: the core and the system commands. The core interprets the inputs and calls the proper programs with the proper input/output options, and the commands are individual, separate programs which are in the directory. There is no distinction between user and system commands, they are all equal in the directory. Any number of the commands may be deleted at any time to reduce memory consumption, though most are not very long. The remainder of this installment will describe the commands.

The minimonitor (mm) command simply jumps to the ROM low level monitor. This allows simple transfer of control back and forth between the different monitors. For this reason, the low level type of instructions which are available in the Amsat monitor are not duplicated in the main operating system. One often used reason for calling the minimonitor is to change baud rate. There is a command in the ROM monitor to do this, though it would be a simple matter to include it in the large monitor as well.

The directory command lists the name, byte count, starting address, and executable address of files in the directory as already described.

The assign (asgn) command creates a directory entry. Typing 'asgn garbage' will create a file named 'garbage'. The

assign command requests the entries for the directory.

The remove (rm) command removes files from the directory. 'rm file1 file2 file3' removes those files.

The move (mv) command changes a file name. 'mv file1 file2' changes 'file1' to the name 'file2'.

The copy (cp) command copies files. The files may be either ASCII or binary files. The byte count is used to determine the length of the file, so it must be correct before the copy command is used. 'cp file1 file2' copies 'file1' into 'file2'. The copy command is smart enough to always copy in the proper direction. If a file is being copied to another location which will partially overwrite the original file, the copy command will check first and either copy from the bottom up or from the top down, which ever is required. This feature is useful for moving files around in memory. A temporary intermediate file must be created to do so.

The case (case) command converts ASCII files to all upper or lower case. 'case u file1 file2' puts 'file1' and 'file2' to the standard output in upper case. 'case l file' does the same with lower case. Note that this neither changes the file nor creates a new file. That can be done with the universal input/output diverters.

The concatenate (cat) command concatenates ASCII files to the standard output. 'cat file1 file2 file3' prints those files, but could be used to concatenate those files together with the output diversions. Cat is the normal way to list a file. ASCII files are ended with a byte '01'. This is true throughout the entire system.

The update (upd) command updates the byte count in the directory for ASCII files.

The compare (cmp) command compares ASCII or binary files. 'cmp file1 file2' compares the two files. The byte count must be correct. If the first file is an executable file, then any differences will be noted with the format 'Differ: Char 2345, 5678' if those are the addresses of each file where the first difference is found. If the first named file is a nonexecutable (ASCII) file, then the first difference will be noted with the format 'Differ: Char 5, Line 122'. This is useful for checking

with the text editor. If the files agree totally with each other for their byte count in length, then the output 'identical' is printed.

The word count (wc) command gives the number of bytes, words, and lines of an ASCII file in decimal. For instance 'wc art' tells me that as of this point in my writing this article, the file is 26843 bytes long, contains 4672 words, and is 575 lines long.

The set teletype (stty) command sets the standard input/output options. As I mentioned in the hardware section, the computer has two independent serial interfaces. It also has a printer. To control these possible modes of input/output, the stty command has various options. 'stty outp' turns on the printer. That is in parallel with whatever else is currently specified. Normally that is the console video terminal. 'stty -outp' turns off output to the printer. 'stty out1' sends output to serial output port #1, and similarly for out2, -out1, and -out2. Input can be taken from either of the two serial ports, and 'stty in1' specifies port 1. Similarly, in2, -in1, and -in2. Also the stty command can enable or disable the echo. The system is full duplex. 'stty echo' and 'stty -echo' would accomplish the obvious.

The return (ret) command permits control to be continued at the point where the program relinquished control to the monitor. That sounds complicated, but it isn't. There is a routine in the operating system which other programs may call to permit a temporary escape to the monitor. For instance, in my version of BASIC, typing 'ESC' tells BASIC to call that routine in the operating system, and the monitor again resumes command. To return to the original program at the same place, the 'ret' command is used. For instance, often in BASIC a program will be set up, and an ESC will be made to use the stty command to turn on the printer, and the 'ret' used to return to BASIC. Note that if the normal exit were used, there would be no means to reenter BASIC without losing the current BASIC program. Another good use would be to divert output to a file. 'ret >= file.out' would return to BASIC, in this case, but start placing output in the file 'file.out'.

The shell (sh) command is an innocent but very powerful command. In itself it does nothing but change the normal resetting of input/output options. Consider anything which can be done on the console keyboard. Any combination of commands can be put into an ASCII file, say 'commandlist'. If then the command 'sh < commandlist' is issued, then the shell program will take input from the file 'commandlist' and execute commands from that file. This permits systems programming simply by creating a file of monitor commands. The commands continue to come from the named commandlist until an 'exit' command is reached. The exit command does nothing but restore the input/output options to the standard input/output.

And last, but not least, is the format (fmt) command. This is the largest and most complex of all the system programs. It is in the real sense a text formatter. Various versions of this are found on the larger timesharing systems. This formatter can fill text, justify the right margin, number pages, create headers on all pages, center and underline lines and words, single or multiple space, indent, and much, much more. A separate description is included of the format program, which I am using to type this manuscript.

There are also various subroutines in the operating system for use with other programs. These include getting hex nibbles, bytes, and words, as well as printing the same. There is a message printing routine, input and output character routines, a routine to input an entire line, one to check if a character has been typed, and a find file routine which locates a file in the directory. There is a conversion program for the Friden justowriter which handles the problem of shifting cases properly as well as doing the character mapping. There are some ASCII characters which are not found on the justowriter, and the printer driver routine prints words describing the ASCII character. For instance, for the greater than symbol, it prints ' GREATER '. Conversions for Baudot or other similar printers would use the same program, and merely by changing the table any desired format could be created. Some of these input/output routines may sound trivial, but the important thing is that all input/output is filtered through the operating system in order

for the various options to be enabled for any other possible programs which run on the system.

Globally throughout the operating system, certain control characters can be used for simple but convenient control. Control 'S' halts output until another character is typed (even another control 'S', so that that character can be used to toggle the output). Control 'O' stops output, but continues processing until a second control 'O' is typed. This skips portions of output which may not be desired to be seen. Control 'D' kills any process. Control returns to the monitor, and on certain programs (like the editor) the program can be resumed by using the return command. Control 'U' dumps any input line completely, and 'rubout' backs up a character and rubs it out.

Well, that is a description of the system. Beginning next month, the schematics of the hardware will be given, followed by printed circuit layouts to complete the hardware description. Subsequently, the various portions of the operating system will be listed with detailed explanations. A sample printout of the system is included, though it is impossible to demonstrate more than a minimal amount without taking many feet of output.

```
 % asgn demo.1
Byte count? 0000
Starting addr? 2000
Executable addr? ffff
% cat demo demo GREATER demmo.1
% cat demo.1
This is a demonstration file.
I will only make it these two lines.
This is a demonstration file.
I will only make it these two lines.
 % wc demo demo.1
demo:
68 Bytes
13 Words
2 Lines
demo.1:
135 Bytes
26 Words
4 Lines
% upd e POWER U
upd demo
% dir demo
                Directory
Name                  Bytes Addr Exad
demo                   0044 1000
% cmp demo demo
Identical
% cmp d POWER U
upd demo.1
% cmp demo.1 demo
Differ: Char 1, Line 3
% cmp basic demo
Differ: Char A980, 1000
% case u demo
THIS IS A DEMONSTRATION FILE.
I WILL ONLY MAKE IT THESE TWO LINES.

% case l demo
this is a demonstration file.
i will only make it these two lines.

% That should be enough for now!
That: not found
% rm demo demo.1
% stty -outp
```