vol = argv[3];
seg = argv[4];
offset = argv[5];
while(strcmp(offset, "0000") != 0)
{
    sprintf(command, "DUMP %s FROM %s:%s:%s
FOR 4 BYTES", node, vol, seg, offset);
    pipein = popen(command, type);
    fscanf(pipein, "%s %s %s %s %s", 
            &data_field[0], &data_field[1], &offset[0], &offset[1]);
    printf("%s\n", data_field);
    pclose(pipein);
}

In this program, the popen statement (in the while loop) creates a process that executes the command previously specified by the sprintf statement as the DUMP command, and sets the pipein variable to point to the file containing the four bytes (in hexadecimal ASCII format) inspected by the DUMP command. Next, the contents of the first two of these bytes are assigned to the data_field variable, which is then printed. Finally, the contents of the second two of the bytes are assigned to the offset variable and become the new value.

Let us give another example of the utility of the Unix shell interpreter—a Unix makefile17 that handles the DO and LOAD commands in order to automate and speed up the development and the downloading of executable files. We assume that several source files, file1.s, file2.s,..., fileN.s are to be assembled, linked, relocated, and loaded on node 1 of a target as two segments, one containing the code and starting from addresses [00, 00, 0000] and the other containing the data and starting from addresses [00, 01, 0000]. The text of the makefile, which is self-explanatory, is:

fileout.out: file1.o file2.o ... fileN.o
    DO fileout.out LINKING file1.o:code, file2.o:code,..., fileN.o:code
    FROM 00:00:0000
    AND file1.o: data, file2.o: data, ... , fileN.o: data
    FROM 00:01:0000
    LOAD fileout.out ON 1

file1.o: file1.s
    as -o file1.o file1.s

file2.o: file2.s
    as -o file2.o file2.s

... 

fileN.o: fileN.s
    as -o fileN.o fileN.s

We have proposed a debugger, Multibug, suitable for distributed target systems. It allows the user to develop software in a host under Unix operating systems, to load executable modules into the target's nodes, and to perform a debugging session with a set of symbolic and interactive commands. The pair {Unix shell, Multibug commands} is similar to a debugging language. Multibug allows the user to maintain complete visibility of the target architecture.

The user can specify a "personalization" of the development environment by defining typed variables. He can assign a symbolic name and a display format to each variable, according to his needs.

Multibug consists of two modules. The first module, HBUG, runs on the host, is written in the C language, is independent of the target, and constitutes 90 percent of Multibug. The second module, NBUG, resides in each of the target's nodes and "virtualizes" each node's physical structure. The NBUG module is quite simple—it is written for a node with a segmented Z8001 microprocessor and uses only 4K bytes. 

Acknowledgments

The work upon which this article was based was supported in part under a grant from the Italian Ministry of Education.