NOTICE:

For support and printed manuals for these products please contact Brian at:

BrianWatson,
"Number Six",
Windmill Walk,
Sutton,
ELY
Cambs
CB6 2NH
ENGLAND

or

brian@spheroid.demon.co.uk

This manual has been reproduced with his permission.

Manual scanned by Paul Collins.
OCRed by Kevin Thacker.

MAXAM 1½ SUPPLEMENTARY DOCUMENTATION

Introduction

MAXAM 1½ has been designed primarily to be used in conjunction with Protext, and is a complete replacement for Maxam. The original Maxam text editor has been removed from the ROM to make room for the new features.

This supplementary documentation details all the new features. The original Maxam manual should be consulted for all other commands. Of course the section in the manual concerned with the editor and the editor menu should be disregarded when rising Maxam 1½.

The new features fall into three categories:

(i) New debugging facilities

These include improved diagnostic information at breakpoints, the ability to use Protext and external commands whilst at a breakpoint, and various external commands to set register values, disassemble, and handle breakpoints.

(ii) Other external commands

Several other external commands have been added, including LOAD and SAVE, MEM, ROMON, HELPR. The existing commands HELP and ROMOFF have been improved. DIF disassembles code to a file.

The most important of these, though, are the commands to assemble the Protext text directly (ASM and TASM).

(iii) New assembler directives

A few extensions have been made to the assembler in the form of directives that can he used within a program. Most usefully, an option has been added to allow an assembly listing to be sent to a file.

How to use Maxam 1½

(a) Assembling from Basic.

This still works as described in the Maxam manual. Note that Protext must be used in program mode. If an attempt is made to assemble a Protext document file, the assembler will give the error 'Bad file'. To ensure this simply load the file into Protest, change to program mode, and save it.

(b) Assembling from Protext. A suggested procedure

Set HIMEM to a value low enough that there is room above it for the machine code, and high enough that there is enough room for the source code in Protext. HIMEM can be set from Protext using the MEM command (see below).

Enter and edit the program using Protext then use ASM to assemble the text. Don't forget to save it – Maxam 1½ cannot prevent the text being lost if the program crashes. Then use J and O to test it. Breakpoints can be assembled into the text using BRK, as described in the Maxam manual, or set after assembling using the new commands explained below.

When a breakpoint is reached after using a J or O command the registers and memory can be examined, and altered if necessary using the commands explained below. Changing the registers is often useful when a bug has been found, enabling the next part of the program to be tested without reassembling.

Assembling the Protext text

ASM

ASM will assemble the program in memory, in the same way that the menu option ‘A’ assembles the Maxam editor text. Simply type the command from Protext command mode.

TASM

Test assemble. This will check the text for errors but will ignore undefined symbol errors.

This can save a lot of time when assembling a large program consisting of several source files. It is frustrating to assemble a large prograrn only to find a jump out of range or a typing error. After editing one of the source files use TASM first. ASM would undoubtedly give many undefined symbols, but TASM just lists all the errors that are, contained within the file.

O (resume) is used to resume from a breakpoint point. There is a very important difference between J and O. The command J should be used ONLY to start the execution of a program. It causes a return address to be put on the stack so that if the program reaches its final RET instruction it can return cleanly to Protext or Basic. The 0 command simply jumps to the code without affecting the stack. If J is used instead of O a problem will be caused if the program being debugged has changed the stack pointer since the last breakpoint. In this case the program's stack would become corrupt.

If O is used without a preceding J there will be no problem - it will act as if J was used.

Note: O may seem a strange choice of command but Maxam 1½ is designed to be used from Protext which has used up most single letter commands already.

A note about stacks and breakpoints

Maxam 1½ keeps two stacks - one for its own use, the other for the user program. The user program's stack is stored in Maxam's private workspace and so is quite inviolate.

If a breakpoint is reached but neither J nor O was used to commence execution, the breakpoint information will be displayed but it will not be possible to enter commands. Instead pressing ESC will warm start Basic, any other key will resume execution. The reason for this is that Maxam 1½ does not know what is on the stack and has not switched to the user stack. This may occur if execution is started with CALL (either Basic or Utopia) or by an external command, or from the Maxam (V1) editor menu.

Miscellaneous commands

DI - Disassemble memory (as menu option D)
DIF - Dissassemble memory to file
LI - List memory (as menu option L)
LIF - List memory to file

These four commands will prompt for the parameters if just the name is typed. Alternatively the parameters can he typed on the same line. The syntax is as follows:

DI <start address> <end address>
or DIF <filename> <start address> <end address>

The end address my be omitted, in which case the disassembly or list will continue to to the end of memory, or until ESC is pressed twice.

Example: to disassemble from &9000 to &90FF to a file "PROG.ASM"

DIF PROG.ASM &9000 &9OFF

Breakpoint commands

Up to 8 breakpoints are allowed, plus one temporary breakpoint. The addresses are stored in a table until the J command is given, when the breakpoints are inserted into the text. The temporary breakpoint is used once only, and is cleared automatically when it is reached.

When a breakpoint is hit, it will not be inserted by the next J or 0 command because the code at that address needs to be replaced. It will however remain in the breakpoint table and be used subsequently. It should he noted that if it is required to step round a loop, either two breakpoints should be set, or a single permanent breakpoint assembled into the code.

BR - Set breakpoint
CB - Clear breakpoint
TB - Set temporary breakpoint
DB - Display breakpoints
CAB - Clear all breakpoints
WB - Wait at breakpoint (waits for a key before showing registers)
QB - Quick breakpoints (cancels WB)
BRKOFF - Disable breakpoints - This is useful to test a program without breakpoints if permanent breakpoints are present (i.e. assembled using the BRK instruction), or if the breakpoints will still be needed.
BRKON - Enable breakpoints - Cancels the effect of BRKOFF

Debugging commands

J Jump to code
O Resume execution (go On)

These commands will jump either to the specified address, e.g.

```
J &100
```

or, if no address is given to the current PC value. This can be checked by typing REGS or PC.

Any breakpoints that are set will be inserted (except at the current PC address, see below) unless breakpoints have been disabled with BRKOFF. If a breakpoint is hit, the screen will display the registers and the memory pointed to, as well as a disassembly of the next few instructions. It will then be possible to use any Maxam 1½ or Protext commands before using O to resume execution.

New Assembler commands and directives

LIST F < filename >

The LIST drective has been extended to allow the assembly listing to be sent to a specified file. This is most useful in conjunction with NOLIST to obtain a file of error messages. Users of Promerge Plus can then load the source code and error files together, and swap back and forth with CTRL-Y.

Example: LIST F "prog.pm"
NOLIST

Important note: due to the limitations of Amsdos it is not possible to use LIST F and WRITE together. If this is attempted the file opened first will be abandoned.

STR < list of strings >

STR is similar to BYTE and TEXT with the option that it will only take a list of strings and the last character in each string has the top bit set. This is useful when printing a string character by character, as you then only need test if each character has its top bit set to know when you've reached the end of the string. The AMSDOS firmware stores command names in this way.

STOP

The STOP command causes reading from the file containing the STOP directive to terminate and return to assemble the remainder of the program in memory. The END command would abort the assembly completely.

PRINT

This command has been extended to allow variables to be included. To print the value of a variable in hexadecimal precede it with “$” (decimal) or “&” (hex)

Example: PRINT “The code ends at &endprog and is is $len bytes long”.

Register commands

REGS	Display registers	**HL**	Set HL
AA	Set A register	**IX**	Set IX
AF	Set A and F	**IY**	Set IY
BC	Set BC	**PC**	Set PC
DE	Set DE	**SP**	Set SP

With each of the commands to set registers type the command name followed by the value required.

Examples: AA 47
|HL,&ABCE (syntax required from BASIC)

MEM - Set HIMEM (as BASIC command MEMORY)

Example: MEM &7FFF

LOAD – load a file into memory

Syntax: LOAD <filename> (<load address>)

This LOAD command has much greater use than the equivalent BASIC command, which only operates with binary files, and only allows files to be loaded at an address greater than HIMEM. Here there is no restriction on either file type or address, and the command is of particular use for loading machine code files.

The file is loaded at the specified address, or, if the address is omitted, to the load address from the file header. In the case of ASCII files there is no header so an error message will be given if no load aderress is specified. The load address is not checked in any way, so ill-chosen addresses may cause a system crash. LOAD closes the input file, if there is one.

Examples:
1. |LOAD,”binary”
Load file called “binary” at the address in the header.
2. |LOAD,"ascii”,&3000
Load file “ascii” at &3000.

SAVE - save block of memory as binary file

Syntax: SAVE <filename > <start address> <length> (<load addr>)

Any block of memory can be saved, and is specified as in the BASIC command SAVE by start address and length. The fourth parameter is optional and specifies the entry address for machine code programs. The fifth (optional) parameter is an addition to BASIC and allows the setting of a load address different to the address from whicb the file was saved. This latter option is only available to disc users because the cassette firmware does not allow the load address to be set. SAVE closes the output file, if there is one.

HELPR - list RSX commands

This lists all external cummands provided by RSXs that have been loaded from tape, in the same way that HELP,n lists external commands provided by background ROMs.