DRAFT FOR APPROVAL
PUBLISHED: 3/2/66

Identification

GEBUG
D.B. Wagner

GEBUG was written in the summer of 1965 by S. Kidd from specifications by R. Fenichel. It is now being maintained by D.B. Wagner.

Introduction

GEBUG is an interactive symbolic post-mortem dumper, used in debugging 645 programs run under the 6.36 System. At the termination of a 6.36 run, a core image file and possibly some BSA symbol table files are returned to the user's CTSS file directory. GEBUG uses these files to provide 'peeks' in various formats at the condition of the simulated 645 core storage at termination. In conception but not in details it resembles FAPDBG.

It seems to be a general property of large interactive programs that such a program is difficult to describe but serves as its own best tutorial aid. The user wishing to use GEBUG is advised to skim this document, primarily the section "Summary of Requests," then start playing with the program. Once a feeling has been obtained for the general philosophy of GEBUG, a closer reading of this document might be worthwhile.

Usage

To use GEBUG, the user links to GEBUG SAVED in the public file, M1416 CMFL04. GEBUG is entered by the command,

```
     RESUME GEBUG runnam
```

Where runnam is the run name specified to the merge editor (it is the primary name of the dump file).

After a certain amount of initialization, GEBUG types 'GO AHEAD' and begins reading requests from the console. If a request begins with a "/", it is a "control" request, otherwise a "peek" request.

Requests are normally typed one per line, with program response following immediately. Several requests may be placed on the same line, however, separated by semicolons, and no program action will take place until the carriage return is typed.

Location Control Requests

These control requests specify how symbolic location references are to be interpreted. Two concepts are important here: "current segment" and "current stack frame". The first is controlled by the /segment request, the second by the /spu and /spd ("stack pointer up" and "stack pointer down") requests.

BSA produces a symbol table file for each segment giving the values of all the symbols defined in the source program for that segment. The symbols are divided into three categories: text (ordinary) symbols, stack symbols, and linkage symbols. The request

    /segment alpha

makes alpha the current segment (more about this later) and causes the symbol table file alpha DEBUG to be loaded so that these symbols are available for the interpretation of input expressions and for the printing of instructions symbolically (instruction output format).

GEBUG keeps a copy of the active registers, which may be interrogated with the /registers request. The value of the base pair sb-sp determines "current stack frame" for location references. Several requests have been provided for changing this copy of the active registers, the most important of which are the two described below. The request

    /spu

moves the stack pointer up (the direction of increasing addresses, or the direction of subroutine calls). That is, it loads sb-sp from sp|18, then loads the copy of the active registers contained in the new stack frame. This copy of the active registers is the contents at the time of the last call out from this stack position.

    /spd

moves the stack pointer down (the direction of decreasing addresses, or the direction of subroutine returns). That is, it loads sb-sp from sp|16 and loads the active registers from the copy contained in the new stack frame. (See BD.7.02 for details of the stack.)

The 'peek' request takes either of the forms

    loc1

    loc1 to loc2

where loc1 and loc2 are expressions which may include text or stack symbols, octal offsets, and the symbol "*" (which always refers to the last location printed by a peek request). Some examples of such expressions are:

```
77

SYM +1 - SYM2 + 6 - 1

* + 23
```

The _value_ of such an expression is not enough to give the full intent of the user. The expression must be _interpreted_ as to whether it refers to a segment or an absolute address or the stack, etc. If such an expression contains a stack symbol, it is taken to refer to the current stack frame, and if it contains a text symbol, it refers to the current segment. If the expression contains nothing but octal numbers it is interpreted as follows: Normally it refers to the current segment, but the requests

/_stack_

/_absolute_

/_dump_

set purely octal expressions to refer to the current stack frame, absolute locations in simulated 645 storage, or directly to locations in the core dump file, respectively. The /_segment_ request always resets this option, and the /_segment_ request without an argument does nothing else but reset this option.

If an "*" occurs in an expression, the expression is interpreted in the same manner as the last expression evaluated.

The contents of the specified locations in the "peek" request are printed out one per line in each of the formats specified by the preceding /_format_ request. This request takes the form,

/_format_ f1 f2 f3 . . . fn (abbreviation /_f_)

where _f1_, _f2_, etc. are format names. The possible format names are _octal_, _floating_, _decimal_, _ascii_, _instruction_, and _indirect_ (abbreviations _o_, _fl_, _d_, _a_, _i_, and _ind_). All but _indirect_ are self-explanatory. All are described in some detail later. If no format request has been given, "/format octal" is assumed.

Other Requests

The existence of the 645 instructions _sreg_, _lreg_, _stb_, and _ldb_ (store registers, load registers, store bases, and load bases) guarantees that whenever a program saves machine conditions for one reason or another, a great deal of information useful in debugging will be stored besides that which was of immediate interest. The GEBUG requests described below are designed to take advantage of this fact.

/*lreg* loc

/*ldb* loc

/*ldi* loc

/*rtcd* loc

cause the program registers, base registers, indicator register, or indicator register and instruction counter, respectively, to be loaded from *loc*, under the assumption that this information was stored there by an *sreg*, *stb*, *sti*, or *stcd* instruction. *loc* is an expression interpreted in the same way as in the peek request.

The request

/*termin*

resets GEBUG's copy of the active registers to the contents at termination time.

Since /*ldb* and /*termin* change the pair sb-sp, they may change the current stack frame.

The contents of GEBUG'S copy of the active registers (as modified by /*spu*, /*spd*, /*lreg*, etc.) is interrogated with the request

/*registers* list (abbreviation /*reg*)

where "*list*" is a list of code words, separated by blanks. The code words are

eaq	the double-precision floating-point number determined by the E, A, and Q registers. Also the E-register.
a	the A-register
q	the Q-register
ir	the indicator register
ic	the instruction counter
dbr	the descriptor base register
pbr	the procedure base register

tr	the timer register
brs	the base registers
xrs	the index registers

If no list is specified, all this information will be printed.

The request

/survey

prints a survey of all segments in the descriptor segment, with all relevant information about each.

Besides the automatic symbol table loading by the /segment request, two less important control requests are available for manipulating symbol tables.

/lsym alpha

causes the symbol table file alpha DEBUG to be loaded, regardless of what segment is current.

/csym

causes the symbol table to be cleared.

The request

/read alpha

causes GEBUG to stop reading from the console and to read instead from the file alpha GEBUG which was created using the TYPSET command.

The request

/xecom command

causes GEBUG to call the library subroutine XECOM to execute the CTSS command specified.

The request

/timer

causes GEBUG to begin printing execution times after each request. This is expected to be helpful in "tuning" GEBUG's I/O, but will not be of great interest to the user.

A "/" followed by a format name followed by any valid GEBUG request causes the format list to be temporarily set to that one format. The request is executed, and then the format list is reset to its previous status. For example,

```
     /a   27   to   36
```

causes the contents of locations 27 to 36 to be printed in the ascii output format.

Output Formats

octal (abbreviation o)

Identical to the Ø12 format of (IOH).

floating (abbreviation fl)

Identical to the E15.8 format of (IOH).

decimal (abbreviation d)

Identical to the I12 format of (IOH).

ascii (abbreviation a)

This format interprets a word as four 7-bit ASCII characters in 9-bit fields. The characters are printed using the escape conventions appropriate to the console being used. The names of control characters are printed, e.g., (000)8 is printed as "NUL". If the character is illegal (high order bits non-zero) the escape convention for an octal specification is used.

instruction (abbreviation i)

The word is interpreted as a 645 instruction and is printed in a pseudo-BSA format. If a symbol table is available, it is used to print the address symbolically as a symbol plus octal offset.

indirect-type (abbreviation ind)

where type is the BSA mnemonic for some 645 indirection modifier (i, id, di, sc, ci, ad, sd, idc, dic, or *) interprets the word as an indirect word referred to by a 645 instruction with the indicated indirect modifier. The word is printed using the BSA pseudo-operations ZERO, TALLY, TALLYB, TALLYC, TALLYD. This format is (hopefully) convenient in working with the specified kind of indirection. If type is not specified, it is taken to be the last indirect modifier seen by the instruction format.

Summary of Requests

loc1

loc1 to loc2

Print the contents of the indicated locations in formats previously specified to the /format request. loc1 and loc2 are symbolic expressions involving text and stack symbols, octal offsets, and "*".

/format list (abbreviation /f)

List is a list of formats indicating the formats in which output is to be printed. Permitted formats are octal, floating, decimal, ascii, instruction, and indirect (abbreviations o, fl, d, a, i, and ind).

/segment alpha (abbreviation /s)

Set the current segment to alpha and load its symbol table if one exists.

/spu
/spd

Move the stack pointer up or down one frame.

/stack

Set pure octal expressions to refer to the current stack frame.

/absolute (abbreviation /abs)

Set purely octal expressions to refer to absolute locations in simulated 645 memory.

/dump

Set purely octal expressions to refer to words in the dump file.

/lreg loc

/ldb loc

/ldi loc

/rtcd loc

Set GEBUG's copy of the active registers as if a 645 instruction sreg, stb, sti, or stcd had stored information in loc.

/termin

Set GEBUG's idea of the active registers to their termination-time contents.

/registers list

Prints the contents of the active registers. If list is a list of code words, separated by blanks, only those registers specified are printed. Permissible code words are eaq, a, q, ir, ic, dbr, pbr, tr, brs, xrs.

/survey

Print a survey of all segments in the descriptor segment.

/lsym alpha

Load the symbol table file 'alpha DEBUG'.

/csym

Clear the symbol table.

/read alpha

Causes requests to be read from the 12-bit mode file 'alpha GEBUG' instead of the console.

~~/xecom command~~

~~Causes the specified CTSS command to be executed, control eventually returning to GEBUG~~.

/timer

Start printing execution times after each request.

Possible additions to GEBUG

One of the purposes of GEBUG is to serve as a model for an interactive debugger for the full-blown MULTICS system, and it will be worthwhile to put some effort into various kinds of improvements in order to get a feeling for the specific aids which are useful in debugging 645 programs. There is however always some danger that a program which is changed too much may become Baroque, existing for its own sake, with dangling loops and pendants serving only to glorify its creator. This would be forgivable in a shortlived program like GEBUG, but only if it leads to a clean debugger for MULTICS.

What follows is a collection of suggestions for possible additions. Implementation varies from easy to frightfully difficult, and notes have been appended to some indicating the amount of work required. The first two are the remaining features described in the original document (R. Fenichel, BE.5.14, 8/06/65) but never implemented.

1. The /word request, which takes a word specified in some named format and prints it out in all the others.

2. The extern output format, which prints a word as an instruction just as in the instruction format, but translates any linkage information present and prints the address as something like "<segnam>|[extsym]±exp". It is becoming evident that 'instruction' formats in general are not going to be as useful in 645 debugging as they are in 7090 debugging, since procedures are normally pure. Hence, even though the extern format would be reasonably easy to code, it will not be provided unless there is some demand for it.

3. An "effective address" output format, which chases down indirections and prints the actual effective address of an instruction word as something like "<segnam>|exp". The overwhelming variety of address modifications available in the 645 will probably lead to many interesting bugs, and it seems clear that this would be a worthwhile feature for a 645 debugging aid. Part of its usefulness would be nullified by the fact that active registers change fairly quickly, so that the calculation of an effective address involving indexing is liable to be worthless. The addition of the /ireg, /ldb, etc. requests and the stack requests was made in part to alleviate this problem, but the usefulness of these requests remains to be judged.

4. A RUNCOM-like macro facility. This would be fairly easy to implement if anyone shows any interest in it.

5. A "search" feature, which in its simplest form could search for a word in core with specified bits having a specified content. A more advanced form might be:

```
     /search   segment  sym=loc1 loc2 conditional expression
```

where the conditional expression could be as complex as that of PL/1 or considerably simpler, but certainly includes the functions C() (contents of) and EA() (effective address of). The request would be very easy to code for GEBUG except for the evaluation of the conditional expression, which will be as easy to code as it is restricted.

6. Extension of location expressions to include linkage symbols. Precisely what interpretation is to be given to an expression containing a linkage symbol remains to be worked out.

7. Extension of the instruction output format to include checking for errors such as an rpd instruction occurring in an even location or an lreg instruction having an address not divisible by 8. This is easy enough to do but if the assembler is at all worthwhile should not be necessary. Again pure procedures save the day.

8. Requests which allow the user to make full interactive use of whatever sort of "snapshot dumper" eventually becomes available for 6.36. Presumably two requests would be provided: /forward and /back. These would move GEBUG's idea of machine conditions (both active registers and contents of core) to respectively the next or the previous snapshot. Then all of GEBUG's essentially static facilities could be applied to machine conditions as they were when that snapshot was taken. The 7090 end of snapshot dumping will be quite easy, but the GE side of it is not at all easy, and considerable thought will have to be given to this whole area.

9. So far we have not considered at all what sort of facilities will be useful in debugging 6.36 EPL programs. Certainly as a minimum it should be possible to request the value of a variable, this value to be printed either in UNSPEC format or in the format indicated by the attributes declared for the variable in the source program. Beyond this, everything is purely in the thinking stage.