RTF

BUGS

Brown University Graphics System

FUDD Debugging Package[1]

A Users' Guide

Russell W. Burns

The Brown University Graphics Project

Division of Applied Mathematics

Box F

Brown University

Providence, Rhode Island 02912

January 26, 1977

---

[1]This research is being supported by the National Science Foundation Grant GJ-28401X, the Office of Naval Research, Contract N00014-67-A-0191-0023, and the Brown University Division of Applied Mathematics; Principal Investigator Andries van Dam.

TABLE OF CONTENTS

BUGS: "Eh, what's up Doc?"

FUDD: "You're a shrewd character Mister Rabbit, but I've got you now!"

1 FACILITIES OVERVIEW

The GMS command FUDD[2] is the Level 1 interactive debugger for the BUGS multiprocessing system. FUDD is not a resident routine during normal execution. Only a small root segment is active in the system at all times. The main portion of FUDD is on the disk. Whenever an undefined interrupt occurs, the root module fetches the main portion of FUDD from the disk and starts its execution. FUDD gives the user the ability to display and store the five different storage types available on the system and to control execution of his program.

The syntax of the FUDD command language is tailored to allow the user access to all storage on the system. Since the system contains two distinct and powerful CPU's, the origin system (appropriate to a simplex operation) has been scrapped, but the useful parts have been kept. Expressions may be used for addresses and lengths, and a programmer may sit down to debug his program without needing a pencil to subtract and add addresses etc. by hand. That is, after all, what computers (usually) are good at.

The BREAK and CHECK POINT facilities allow the user to have up to sixteen break points and/or check points active at any given time. It is also posible for the user to include checkpoints in his program so that he or she can display or modify values during execution. The full capabilities (within core storage limits) of GMS are available to the user while under the control of FUDD. Some of the GMS commands are available directly (QUERY, MODMAP, LOADMOD, DELETE, and MODZAP) and the rest may be accessed through the GMS command.

---

[2] Name courtesy of Susan Beckley.

2 COMMAND SYNTAX

The commands consist of a command name, which may be abbreviated, followed by one or more parameters, separated by blanks. There are three general types of parameters, the first of these types being a literal. Literals are used for label names (<label>) and CPU identifiers (<cpu-id>). The only valid <cpu-id>'s are 'A' and 'B'. Any character string up to eight characters, beginning with an alphabetic followed by letters and numbers may be used as a label.

The second type of parameter is an address (<address>). Addresses have both a value and a type. The type is determined by the first value in the address string. These types are A-CPU register, B-CPU register, Vector General register, Local storage, and Main storage. The rules for combination of these storage types are not rigorously defined, but seem to be convenient for the sort of things which the normal (and even abnormal) user might want to do while debugging his program. Any complaints about these rules should be written on the walls of the Faunce House men's or ladies' room.

The A-CPU registers are specified by 'R', 'A', or 'AR' followed by a decimal or hexadecimal number. B-CPU registers are specified by 'B' or 'BR' followed by a number, and the Vector General registers are indicated by 'V' or 'VR' followed by a number. Local storage is specified by 'L.' followed by an expression. Unless a register is specified first in an expression, its contents rather than its number are used in computing the final address.

Both decimal and hexadecimal constants may be used in address expressions. Hexadecimal constants are specified by prefixing '/' to the number. The first non-valid character is treated as the end of the number. The parameter scan continues from that point, without demanding an operator.

Any string not recognizable as one of these types is considered to be a label. Labels have three methods of definition; first, the external names of all loaded modules are automatically entered in the label table; second, upon each entry FUDD automatically defines the labels 'A' and 'B' to the beginning of the A-CPU most currently loaded routine, and to the start of the current B-CPU procedure respectively; third, labels may be added to the label table through the use of the DEFINE or TEST commands. Labels and constants always have the type main storage, and an address expression starting with these will

define a main storage address. Labels which are not first in an expression must be preceeded by operator. The valid operators for expression are '+' and '-', with '+' being the default. Since the scan continues after a number with the next character regardless of whether or not there is an operator present, many common expressions may be abbreviated.

The 'L.' storage designation must always be first in a address string, and may not appear in any other conjunction.

Examples of valid addresses:

R1 --> A-CPU register one
BR4 --> B-CPU register four
A --> address of A-CPU routine
L./43 --> local storage word X'43'
4R1 --> four past the contents of register one
B+4 --> four bytes past the start of the main 'B' procedure.

The third type of parameter is the length (<length>) or data (<data>) type parameter. These may be specified in the same manner as addresses, with the exception that the contents rather than the address of registers always are used. Lengths are always considered to be in bytes.

3 ENTERING FUDD

FUDD may be invoked in many ways, but only two are totally under the control of the user. The first of these is to enter the FUDD command to GMS. FUDD will respond with the message 'FUDD HERE ...' and request a command. The second method, which should be used only when the first is not posible, is to depress the interrupt key on the 'A' panel. FUDD will respond with the message:

A:I/O INTERRUPT-(2531)-PANEL AT xxxx.

Where 'xxxx' is the address of the currently executing routine.

For debugging routines which use GETMAX, it is advisable to enter the FUDD command before starting the program. This will insure that there will be sufficient core available for FUDD to be loaded.

Once invoked, FUDD will remain in core until the next IPL, to allow the user to use FUDD during his program test phase, and to define global labels which will remain in force until IPL. A production program need never compete with the debugger for resources, but test sessions should begin with the use of the FUDD command. If FUDD initialization fails, the machine will enter a disabled wait state, with X'FODD' in the upper lights, the name of the event which caused entry to FUDD in the lower lights, and the Program Counter set tc the address where the event occurred.

4 FUDD COMMANDS

The FUDD commands are here divided into five major areas: data manipulation commands, program manipulation commands, execution control, symbolic debugging commands, and system commands. In each section, the commands will be listed in alphabetical order, preceded by the syntactical definition of the command. The command name will be expressed in single quote marks, with the shortest acceptable form underscored. Brackets around parameters indicate that they are optional.

4.1 DATA MANIPULATION COMMANDS

Address operands for DISPLAY, STORE, and XFER are rounded down to the nearest halfword before the data is accessed. Lengths are always rounded up to the nearest halfword.

'DISPLAY' <ADDRESS> [<LENGTH> [<TYPE>]]

The specified location or register is displayed at the terminal, for the specified length. The default length is two bytes. The <type> parameter may take the value 'X' (default), which displays the data in hexadecimal form; it may be 'D', which displays in decimal; if 'T' is specified, the character translation as well as the hexadecimal is printed; and if 'F' is specified, the number is printed as a signed decimal fraction.

'STORE' <ADDRESS> <DATA> [<DATA> [<DATA> [<DATA>]]]

The specified two byte data fields are placed in the specified location, and in accending locations if more than one data field is specified. Up to four data fields may be specified, delimited by blanks.

'XFER' <ADDRESS-1> <ADDRESS-2> [<LENGTH>]

Halfwords of data are transfered from <address-2> to <address-1> for the length specified (default two bytes). To insure against accidental disruption of the data in the machine, the maximum length which can be XFER'ed is 256 bytes.

4.2 PROGRAM MANIPULATION COMMANDS

'DELETE' <MODULE>

The DELETE command operation is identical to the GMS DELETE command.

'LOADMOD' <MODULE>

The LOADMOD command operation is identical to the GMS LOADMOD command.

'MAP' <MODULE> [<START>]

The MAP command is identical to the GMS MODMAP command.

'ZAP' <MODULE> [<CSECT>]

The ZAP command is identical to the GMS MODZAP command.

4.3 EXECUTION CONTROL

'BREAKPT' <ADDRESS>

'BREAKPT' <breakpoint-id> 'OFF'

The BREAKPT command causes FUDD to break execution of the program at the specified address. A break point identifier is associated with this address which is printed after the break point has been set. If 'OFF' is specified, the break point identified by the <breakpoint-id> is removed. When a break point is executed by a problem program, FUDD will be entered and the user may modify storage or his registers. If a BREAK supervisor call (SVC 244) is assembled into a program, it will act as a check point, allowing the user to display or modify data in his program and continue.

'CHECKPT' <ADDRESS> [<CPU-ID>]

'CHECKPT' <breakpoint-id> 'OFF'

A check point will be set at the specified address. The default <cpu-id> will be the 'A'. Different instruction lengths on the two machines make it neccessary to specify which machine to prepare for. Although breakpoints are removed when executed, checkpoints must be explicitly removed.

Hints:

If checkpoints or breakpoints are to be placed on sequentially executed instructions, they should be specified in the order they will be executed.

Checkpoints should not be placed on branch instuctions or any others that modify the program counter. They may be placed on CALL or BAL instructions if the routine called will always return to the intruction following the CALL or BAL.

'GO'

Execution is re-started in both CPU's, and FUDD goes into a dormant state. Execution is continued where it was interrupted, unless the user has modified the META 4A's PC or the META 4B's PBR/PDR.

'HALT' <CPU_ID>

The HALT command causes all execution to halt in the specified CPU, except for that neccessary for FUDD to operate.

'IPL'

The IPL command causes FUDD to force an Initial Program Load of the system.

4.4 SYMBOLIC DEBUGGING COMMANDS

'DEFINE' <LABEL> [<ADDRESS>]

The DEFINE command allows the user to add, change and delete entries in the FUDD symbol table. If no value is specified for the label, it is deleted. If the value is specified, that is made the new value of the label, and the old value is printed out if any.

'TEST' <MODULE> ...

The TEST command may be used to load routines to be tested and to define all external labels in those routines in the symbol table. If the name(s) given is already defined by either being an active program or through use of the DEFINE command, the routine will be treated as though it were already loaded, and the symbol table value will be used as the starting address for resolving the displacements within the

module (e.g. to define the proper labels for LEVEL0, define the label LEVEL0 as being location 0, and then TEST LEVEL0).

'WHERE' <ADDRESS> [<TYPE> [<ADDRESS-2>]]

The WHERE command is used to display where an address lies relative to any other address. The default type is relative, which will display where the address is relative to the appropriate module if possible. If 'A' is specified for the <type>, the absolute value in hexadecimal of the address will be printed. If <type> 'R' and a second address (<address-2>) are specified, the displacement from <address> to <address-2> will be printed as a signed hexadecimal value.

4.5 SYSTEM COMMANDS

'FILEBUG'

The FILEBUG command causes FUDD to call FILEBUG, the Doctor Memory file debugger (vide the Doctor Memory manual for details).

'GMS' <GMS-COMMAND-STRING>

The GMS command allow access to any GMS command not already defined by FUDD. 'GMS' must merely be appended to the GMS command.

'Q' <WHAT> <HOW>

The operation of the Q command is identical to the GMS QUERY command, althcugh most of the operands are more meaningful when entered during debugging.

5 SAMPLE TERMINAL SESSION

The following is a sample session, with the user's entries in lower case and the FUDD responses in upper. Meta-comments are in brackets.

```
> 1 fudd                          [enter FUDD command]
FUDD HERE ...
1 program                         [LOADMOD PROGRAM]
3C00
-define a program                 [define label 'A']
A       AT 3C00
-break a+/10                      [put break point at A+X'10']
BREAK POINT 0000
-wh a+/10                         [verify location of break point]
PROGRAM+/10
-go                               [return to GMS]
>program                         [start program]
A:BREAK POINT 0000 AT 3C10
-where r1                         [inquire as to location]
PROGRAM+/10
-display r5 2 t                   [display R5 in hex and character]
C2FF <B.>
-st r5 /c2c2                      [store X'C2C2' in R5]
-st r6 r5                         [put contents of R5 in R6]
-d r5 4 t                         [display R5 and R6 in hex and
                                  character]
C2C2 C2C2 <BBBB>
-go                               [continue execution]
```