COMPUTATION CENTER
Massachusetts Institute of Technology
Cambridge 39, Massachusetts

TO: All Programmers.

FROM: R. H. Campbell.

DATE: April, 1965.

SUBJECT: FAPDBG, a symbolic debugging aid.

This memo is an attempt to describe FAPDBG, a symbolic debugging aid for FAP subprograms. It supersedes memos CC-216, CC-216 supplement 1, and CC-216 supplement 2. The FAPDBG subprogram is a primitive version, produced primarily in order to experiment with the possible typing conventions and formats. Suggestions and complaints will be of great assistance in planning future changes and additions to the debugging system.

FAPDBG acts upon requests typed by the user on the console and performs such functions as examining and typing or changing the contents of specified registers, and allowing a subprogram to be run in controlled segments. The requests take the form of a single-letter request name followed by arguments, all separated by blanks. A spurious blank may precede the request, but none may follow it. A blank is a string of any (non-zero) number of spaces or tabulations. Any number of requests may be concatenated on one line merely by typing an apostrophe or an equal sign between the successive requests. The advantages of concatenation lie in the fact that the subprogram will have to be brought into core less often and will generate more output each time in.

The available requests may be divided into the four classes of set up, register modification and examination, subprogram control, and FAPDBG control. Each of these classes will be considered separately.

Set up Requests

The set up requests permit the user to inform FAPDBG of the symbols used in the subprogram he wishes to debug. They are load address, symbol table, work, and equals.

1. L: load address.

   a. L EP

causes FAPDBG to search the loading table for the subprogram with an entry point name EP. When FAPDBG finds the origin of the subprogram, FAPDBG will type it out and save it for use in relocating the symbol table. If the entry can not be found, FAPDBG will inform the user of this fact and return to process the next request.

2. T: symbol table.

a. T FN

causes FAPDBG to read and relocate the symbols in file FN SYMTB by a relocation constant equal to the last origin typed out. If such a file exists, FAPDBG will read it, add the subprogram origin to each symbol value, and merge the symbols into its symbol table. Note that the SYMTB file contains no relocation information; thus absolute and common symbols will also be modified by the subprogram's origin. This procedure will produce improper values for all absolute symbols and for common symbols in all but the first-loaded subprogram. FAPDBG will signal successful completion by typing "SYMBOLS LOADED." if the file can not be found, FAPDBG will inform the user of this fact and return to process another request. If the symbol table becomes full, FAPDBG will stop reading the file and inform the user of the last symbol inserted in the table. This information may be of some use since the symbols in the SYMTB file are in alphabetical order.

b. T

causes FAPDBG to delete all the user's symbols from the symbol table.

3. W: work.

a. W EP FN

is identical to the two requests L EP'T FN, i.e. it finds the origin of the subprogram with an entry point EP, then reads and relocates the symbols in file FN SYMTB. If the subprogram cannot be found, no attempt will be made to read the symbol table. Having given a work request for one subprogram does not imply inability to refer to locations in other subprograms. Work is merely a combination of the two requests load address and symbol table.

b. W EP

is a brief form, permissible when FN and EP are the same. It is equivalent to W EP EP, or L EP'T EP.

4. E: equals.

a. E FE FS

permits the user to define or redefine the symbol FS to have the value of the expression FE. If the symbol table is full, an attempt to enter a new symbol will fail and FAPDBG will inform the user of that fact.

Register Examination and Modification

The register examination and modification requests permit the user to examine and change the contents of locations in memory as well as the live registers of the machine. They are floating point look, Hollerith look, full word integer look, FORTRAN integer look, octal look, symbolic look, deposit, compare, signed accumulator, logical accumulator, and storage map.

5. F: floating point look.

a. F LOC1 LOC2

will set the output conversion mode to floating point and type out the contents of the block of locations, LOC1 through LOC2.

b. F LOC

will set the output conversion mode to floating point and type out the contents of the single location, LOC.

c. F

will merely set the output conversion mode to floating point.

6. H: Hollerith look.

a. H LOC1 LOC2

will set the output conversion mode to Hollerith and type out the contents of the block of locations LOC1 through LOC2.

b. H LOC

will set the output conversion mode to Hollerith and type out the contents of the single location, LOC.

c. H

will merely set the output conversion mode to Hollerith.

7. I: full word Integer look.

a. I LOC1 LOC2

will set the output conversion mode to decimal integer and type out the contents of the block of locations, LOC1 through LOC2.

b. I LOC

will set the output conversion mode to decimal integer and type out the contents of the single location, LOC.

c. I

will merely set the output conversion mode to decimal integer.

8. J: FORTRAN integer look.

a. J LOC1 LOC2

will set the output conversion mode to that of a FORTRAN integer and type out the contents of the block of locations, LOC1 through LOC2.

b. J LOC

will set the output conversion mode to that of a FORTRAN integer and type out the contents of the single location, LOC.

c. J

will merely set the output conversion mode to that of a FORTRAN integer.

9. O: Octal look.

a. O LOC1 LOC2

will set the output conversion mode to octal and type out the contents of the block of locations, LOC1 through LOC2.

b. O LOC

will set the output conversion mode to octal and type out the contents of the single location, LOC.

c. O

will merely set the output conversion mode to octal.

10. S: symbolic look.

a. S LOC1 LOC2

will set the output conversion mode to that of a symbolic machine instruction and type out the contents of the block of locations, LOC1 through LOC2. FAPDBG will convert the address, tag, and decrement or count fields of a symbolic machine instruction in the following manner. If a symbol has the same value as the field under consideration, that symbol will be typed. If no symbol has the proper value, the user's symbol which has the closest value will be typed, followed by a plus or minus sign and the necessary correction. The correction will always be typed in octal. If the value of the "best" user's symbol results in a correction whose magnitude is greater than the field under consideration, the field itself will be typed in octal.

b. S LOC

will set the output conversion mode to that of a symbolic machine instruction and type out the contents of the single location LOC.

c. S

will merely set the output conversion mode to that of a symbolic machine instruction.

In addition to the six "look" requests, F, H, I, J, O, and S, one may obtain the contents of any single location in the current output mode merely by typing the location. Of course the first symbol in the location expression must not be a single letter. The contents of location "*+1" may be obtained in the current output mode by typing an empty request (just a carriage return or concatenation character).

11. D: deposit.

a. D LOC FW

will cause the FAP word, FW, to replace the previous contents of the specified location, LOC. This request may be abbreviated by omitting the request name, provided that the location expression does not begin with a single-letter symbol. The FAP word may be a symbolic machine instruction such as CAL ALPHA-10,4 or one of the data generating pseudo-instructions OCT, BCD, FLO, INT (full word decimal integer), of JNT (FORTRAN integer) followed by a blank and one word of data.

A symbolic machine instruction consists of a symbolic operation code, an optional asterisk to indicate indirect

addressing, and an optional variable field in the same format as accepted by FAP, except that all numbers are interpreted as octal and that multiplication and division are not allowed. No blank may intervene between the operation code and the indirect flag; a blank must, however precede the variable field. Note that since the address field is truncated to fifteen bits, the left three bits of the address part of type D instructions (left and right half indicator operations) will be considered by FAPDBG as the tag field, both for input and for output. Thus to insert the instruction

RFT 300105

It is necessary to type

RFT 105,3

The OCT pseudo-instruction accepts a signed or unsigned octal integer of magnitude less than or equal to 377777777777. Thus, to insert the traditional fence, it is necessary to type

OCT -377777777777

The FLO pseudo-instruction accepts a signed or unsigned floating point number with optional decimal point and optional E modifier to denote multiplication by the indicated power of ten. The B modifier is not allowed.

The INT and JNT pseudo-instructions accept signed or unsigned decimal integers of sufficiently small magnitude to fit into the number of bits available (34359738367 for INT and 131071 for JNT).

The BCD pseudo-instruction accepts any string of characters preceding the request terminator (carriage return, apostrophe, or equal sign) and assembles the last six into one word. If fewer than six characters are typed, spaces will be inserted on the left. Note that this pseudo-instruction uses the input line image after FAPDBG has edited and "normalized" it. Therefore a string of spaces and tabulations will be interpreted as a single blank.

12. C: compare or verify.

a. C EP FN

allows the user to find out which registers of the subprogram have been changed from their initial value when loaded. FAPDBG will search the loading table for the subprogram with entry point name EP. FAPDBG will type out the origin of that subprogram and will save it for

relocating file FN BSS. Then it will read and relocate each word in file FN BSS and compare it with core. FAPDBG will type in the current mode all locations and their contents for which there is a discrepancy. The word obtained from the file will be typed first, followed by the word obtained from memory. When file FN BSS has been completely checked against memory, FAPDBG will type "EXAMINATION CONCLUDED." and will return to process another request. This request may be terminated at any point by pushing the interrupt button. FAPDBG will close out the BSS file and will return to process the next request. If the desired file cannot be found, FAPDBG will so inform the user and return to process the next request.

b. C EP

is equivalent to C EP EP. It may be used whenever the file name and an entry name are identical.

13. A: signed accumulator.

a. A FW

places the FAP word FW in the signed accumulator and clears the P and Q bits.

b. A

types out, in the current mode, as set by the previous F, H, I, J, O, or S request, the contents of the signed accumulator, followed by the P and Q bits.

14. K: logical accumulator.

a. K FW

places the FAP word FW in the logical accumulator and clears the S and Q bits

b. K

types out in the current mode, as set by the previous F, H, I, J, O, or S request, the contents of the logical accumulator followed by the S and Q bits.

15. M: storage map.

a. M

will cause a storage map to be typed, with each subprogram listed in the order of loading.

Subprogram Control

The requests which have to do with subprogram control allow the user to run his subprogram in controlled segments. They are break, go, and proceed.

16. B: break.

a. B LOC

conditions FAPDBG to insert a "breakpoint" at location, LOC. FAPDBG will save the location and set an indicator to signal that a breakpoint instruction, specifically a transfer into FAPDBG, is to be inserted into that location. No subprogram modification occurs at this time. An examination of the breakpoint location will reveal its original contents and changing those contents (via a deposit request) will not remove the breakpoint. The breakpoint must not be placed at a subprogram-modified instruction or where it would be used for indirect addressing. Only one breakpoint at a time may be inserted.

b. B

causes the breakpoint to be removed.

17. G: go.

a. G LOC

allows the user to start execution of the subprogram at location, LOC. FAPDBG will examine the breakpoint flag and, if a breakpoint exists, will save the contents of the break location and insert the necessary transfer instruction. It will then restore the machine conditions, and transfer to the specified location.

18. P: proceed.

a. P

allows the user to continue executing his subprogram from the state it was in just before control last entered FAPDBG. Upon encountering the breakpoint transfer instruction, control will be transferred to FAPDBG, which will save the machine conditions and restore the temporarily removed instruction at the break location. FAPDBG will then type "BREAK." and wait for requests.

Proceed will cause FAPDBG to perform all the steps performed by go, except that after restoring the machine conditions, FAPDBG will execute the above-mentioned instruction and transfer to the appropriate location following its location as governed by any skipping which

might occur. If the instruction is location-dependent, namely TSX, STR, STL, or XEC, FAPDBG will interpret it as if it were being executed from its normal location. Thus a breakpoint may be inserted at a subroutine call. A chain of XEC instructions will be interpreted to a maximum depth of ten. A subprogram in operation may be interrupted at any time by pressing the interrupt button.

Internal Operation

The request which controls the internal operation allows the user to return to CTSS. It is quit.

19. Q: quit.

a. Q

returns control to the Time-Sharing Supervisor in such a way that a START command will transfer control to the place in the user's subprogram where it last entered dormant status.

Internal Symbols

The following symbols are permanently defined in FAPDBG as locations where the machine conditions are stored.

$MQ	The multiplier-quotient register.
$SI	The sense indicator register.
$X1	Index register one.
$X2	Index register two.
$X3	Index register three.
$X4	Index register four.
$X5	Index register five.
$X6	Index register six.
$X7	Index register seven.
*	The current location.

This symbol is defined as the last location referred to by either the user or FAPDBG. It is redefined as the location of the next instruction to be executed in the user's subprogram by encountering a breakpoint or by a

manual restart.

$LS Lights and switches.

This location contains the state of the machine conditions in the right-most eight octal digits as listed below; the off status is represented by zero, on status by one. Reading from left to right:

DIGIT	CONDITION
5	Floating point trap mode.
6	Divide check light.
7	Overflow light.
8	Multiple tagging light.
9	Sense light one.
10	Sense light two.
11	Sense light three.
12	Sense light four.

$IC The instruction location counter.

This location contains the address of the next instruction to be executed in the user's subprogram. It is set by encountering a breakpoint or by a manual restart. It is examined by the proceed request in order to determine the location to which to transfer control.

Syntax

The location, address, tag, and decrement parts of a request argument may consist of strings of symbols and octal numbers separated by plus and minus signs to denote the desired algebraic manipulation.

The indicated operations are carried out, the result converted to two's complement form and the right fifteen bits saved (in the case of the tag field of a symbolic instruction, the right three bits are saved). Symbols, which must be defined, may consist of any number of characters, at least one of which must be non-numeric (i.e. not 0 through 7), and none of which may be the special characters; plus sign, minus sign, comma, space, or tabulate. If the number of characters is greater than six,

only the last six will be used. Any string consisting only of the digits 0 through 7 will be considered an octal number of five digits, with left zeros if necessary. If more than five digits are typed, only the last five will be used. The line typed in is scanned from the left and each field is evaluated when encountered. If an undefined symbol is discovered, or a deviation from an understandable format is discovered, an appropriate comment is typed and processing of the request is terminated. If one or more requests cannot be interpreted, any go or proceed requests following them on the same line will be ignored.

Operation

The current version of the BSS loader does not automatically load FAPDBG or any other debugging aid. When the user types one of the in-core debugging commands, the appropriate relocatable subprogram is loaded from the CTSS "debugging" library. This implies that the loader should not be destroyed before issuing any of these commands. If it is feared that the user's subprogram may destroy the loader, FAPDBG may be loaded before the beginning of execution. This procedure should also be used for debugging subprograms which extend the memory bound.

The FAPDBG subprogram is approximately 12400 locations long, octal. This includes storage space for a maximum of 800 symbols, decimal.

FAPDBG handles the interrupt levels in the following way. It always places itself at level one. Any interrupts the user has are moved down to levels two, three, and four. Thus it is always possible to press the interrupt button either to return control to FAPDBG, or to stop its type-out.

The supervisor command

FAPDBG ALPHA

will cause FAPDBG, after announcing its presence, to read requests from the card-image file ALPHA DEBUG. When the file is exhausted, or if the file can not be found, FAPDBG will return to reading requests from the console as usual. The file may be created by use of the ED command.

When FAPDBG is entered, it will announce that fact and type the memory bound if the bound has been changed. The user may then proceed to make requests of FAPDBG. To help get the above descriptions in a concrete example, a "typical session" is presented below. Borrowing a convention used by R. S. Fabry, sample lines will be prefixed by a "U," "S," "F," or "P," to indicate that they are typed by the user, the supervisor, FAPDBG, or the user's subprogram,

respectively. For example, the line

```
F:
```

represents a blank line typed by F.

The hypothetical user has written his own sine routine, assembled and loaded it, and is trying to get it to give correct answers. He first gives the FAPDBG command to inform the loader to load FAPDBG and transfer control to it. FAPDBG acknowledges that control has reached it.

```
U:fapdbg
S:W 1416.0
F:FAPDBG ENTERED.  MEMORY BOUND IS 16730.
F:
```

The user wishes to inform FAPDBG of the symbol definitions to use. He does this by typing a work request which causes FAPDBG to locate the subprogram in core memory and then to read and relocate the symbols in the symbol table file, which the user has named "RHCSIN." FAPDBG informs the user of the two stages of this process.

```
U:w sin rhcsin
F:SIN IS LOADED AT 5212.
F:SYMBOLS LOADED.
F:
```

The next steps the user might take are to insert a breakpoint at the end of his subprogram, set up its input parameters, and transfer control to it. These steps might look as follows; note that FAPDBG types a carriage return just before it tries to read a new line.

```
U:b end
F:
U:a flo 3.14159
F:
U:g sin
F:PROGRAM STARTED.
```

After the "PROGRAM STARTED." comment, the user's subprogram is running. It will run undisturbed until it encounters a breakpoint or enters dormant status for some reason, such as a protection mode violation. From dormant status it is possible (provided the loader has not been destroyed) to restart FAPDBG by typing the command

```
FAPDBG
```

FAPDBG will redefine the current location symbol (*) to be the location of the next instruction to be executed. It is

possible, now, to give a proceed request and continue from the point at which the subprogram entered dormant status.

Let us here suppose, however, that all went reasonably well, and that control reached the breakpoint. FAPDBG will indicate this and then wait for requests.

```
F:BREAK.
F:
```

The user might now reasonably want to examine the output of his subprogram, for example, a floating point number in the accumulator. This he does by changing the output mode to floating point and asking for the contents of the signed accumulator via two concatenated requests.

```
U:f'a
F:$A/              5.6900432     P = 0 Q = 0
F:
```

Unless the user is satisfied with the results, he will probably want to "poke around," examining and changing instructions and data.

```
U:s foo-5
F:FOO-5/           FAD END+17
F:
U:f end+17
F:END+17           1.5707999
F:
U:d foo-5 fsb end+18
F:18 IS NOT DEFINED.
F:
```

This error comment by FAPDBG illustrates an easily overlooked fact. The numbers typed in the variable field of a symbolic instruction, both for input and for output, are absolute octal constants. The string of characters "18" above was interpreted as a symbol which, of course, was undefined.

The user next attempts a concatenation of four requests, but again makes a typing mistake.

```
U:d * fab end+17'a flo 3.14159'g sin'a
F:FAB IS NOT DEFINED.
F:G IGNORED.
F:$A/              3.14159       P = 0 Q = 0
F:
```

Note that the error in the earlier deposit request caused the go request to be ignored. This would not have been the case had the requests been typed on different lines.

Summary of Requests

Below are presented in alphabetical order the request letters, together with a short description of the operation of each request. The numbers in parentheses refer to the section headings above.

A FW

(13a) Replace signed accumulator with FAP word FW. Clear the P and Q bits.

A

(13b) Type in the current mode the contents of the signed accumulator, followed by the P and Q bits.

B LOC

(16a) Insert a breakpoint at location LOC.

B

(16b) Remove the breakpoint.

C EP FN

(12a) Compare the subprogram with an entry name EP with the file FN BSS and type discrepancies in the current mode.

C EP

(12b) The same as C EP EP. May be used when the file name and an entry name are the same.

D LOC FW

(11) Deposit the FAP word FW in location LOC.

E FE FS

(4) Define or redefine the FAP symbol FS to be equal to the FAP expression FE.

F LOC1 LOC2

(5a) Set output conversion mode to floating point and type the contents of the block of storage from LOC1 through LOC2.

F LOC

(5b) Set output conversion mode to floating point and type the contents of the single location LOC.

```
F
```

(5c) Just set output conversion mode to floating point.

```
G LOC
```

(17) Go to location LOC.

```
H LOC1 LOC2
```

(6a) Set output conversion mode to Hollerith and type out the contents of the block of storage from LOC1 through LOC2.

```
H LOC
```

(6b) Set output conversion mode to Hollerith and type the contents of the single location LOC.

```
H
```

(6c) Just set output conversion mode to Hollerith.

```
I LOC1 LOC2
```

(7a) Set output conversion mode to decimal integer and type the contents of the block of storage LOC1 through LOC2.

```
I LOC
```

(7b) Set output conversion mode to decimal integer and type the contents of the single location LOC.

```
I
```

(7c) Just set output conversion mode to decimal integer.

```
J LOC1 LOC2
```

(8a) Set output conversion mode to FORTRAN integer and type the contents of the block of storage LOC1 through LOC2.

```
J LOC
```

(8b) Set the output conversion mode to FORTRAN integer and type the contents of the single location LOC.

```
J
```

(8c) Just set the output conversion mode to FORTRAN integer.

```
K FW
```

(14a) Insert the FAP word FW in the logical accumulator and clear the S and Q bits.

K

(14b) Type in the current mode the contents of the logical accumulator, followed by the S and Q bits.

L EP

(1) Type the load address of the subprogram with an entry name EP.

M

(15) Type a storage map.

N

Unused.

O LOC1 LOC2

(9a) Set the output conversion mode to octal and type the contents of the block of storage LOC1 through LOC2.

O LOC

(9b) Set the output conversion mode to octal and type the contents of the single location LOC.

O

(9c) Just set the output conversion mode to octal.

P

(18) Proceed to the location stored in $IC by the last break or interrupt.

Q

(19) Quit and return to CTSS.

R

Unused.

S LOC1 LOC2

(10a) Set the output conversion mode to symbolic and type the contents of the block of storage LOC1 through LOC2.

S LOC

(10b) Set the output conversion mode to symbolic and type the contents of the single location LOC.

```
S
```

(10c) Just set the output conversion mode to symbolic.

```
T FN
```

(2a) Read and relocate, by the last origin typed out, the symbols in file FN SYMTB.

```
T
```

(2b) Remove all the user's symbols from the symbol table.

```
U
```

Unused.

```
V
```

Unused.

```
W EP FN
```

(3a) Work subprogram with entry name EP whose symbols are in file FN SYMTB. The same as L EP'T FN.

```
W EP
```

(3b) The same as W EP EP or L EP'T EP.

```
X
```

Unused.

```
Y
```

Unused.

```
Z
```

Unused.