TO: MSPM Distribution

FROM: Karolyn Martin

DATE: April 18, 1969

SUBJECT: Revised BY.6.01

This new document on the debugger's request dispatcher reflects the actual implementation as it developed over the past two years. The "do", "end", "if", "then", and "else" requests have not materialized. The dispatching mechanism has changed for efficiency considerations.

Published: 04/18/69

(Supersedes: BY.6.01, 09/28/66)

Identification

Request Dispatcher for Interactive Debugging Aids
dispatch_request, debug_data, parse_scan
M. Wantman, D. B. Wagner

Purpose

The procedures described here read a request from user_input via read_in and "dispatch" the request according to the first word of the request. The procedure that is eventually called gets its parameters via parse_scan so that special characters can be used. Some debugger procedures cannot be called directly in the usual way because they expect to get their arguments in this (unusual) way.

Usage

The call to dispatch_request is made with no arguments:

```
          call dispatch_request;
```

If the first word of the request is recognized as a special debugging request, transfer is made to the appropriate procedure. If the request is not recognized, the shell is called to handle it. Thus any Multics command can be issued from probe.

The called procedure calls parse_scan to get its parameters (if any). The two calls are

```
          call parse_scan$atom (string, break, done);
```

and

```
          call parse_scan$segname (string, break, done);
```

The arguments are declared as follows:

```
        dcl string char(*) varying,
            break char(1),
            done bit(1);
```

Each call returns the next identificable character string from the input request line in the argument "string". The break character that ended the string is returned in "break". If the end of the request line has been reached by this or any previous call "done" is set to "1"b. Otherwise it is "0"b.

The difference between the two entries is that the $atom entry considers the characters "< >" as break characters, while the $segname entry does not. This allows Multics pathnames to be recognized as single entities.

Implementation

dispatch_request calls read_in to get a request line from user_input. The entire line is passed to the scanner by means of the parse_scan$prime entry.

A call to parse_scan$atom returns the first character string on the line. This character string is then compared against the list of known requests which is stored in debug_data. The list is declared

```
        dcl debug_data$names (100) char(32) ext;
```

If a match is found, dispatch_request calls debug_data$dispatch (i), where i is the index of the request in the table of names. debug_data$dispatch makes a direct transfer to the appropriate procedure, and does not make a stack frame for itself.

When the called procedure has finished and returned to dispatch_request, dispatch_request returns to probe. probe checks to see if the last request was "quit". If not, it calls dispatch_request again.

If no match is found and the request is not recognized, dispatch_request directs output back to the user's console if it had been going to a segment (thus setting the bit count in the branch) and calls the shell with the input line.

The procedure debug_data is divided into 4 sections. These are

1. Table of probe requests
2. Transfer vector
3. Procedure to transfer through the transfer vector
4. External storage shared by all debugger procedures.

1. These are stored as 32-character strings generated by the ACI pseudo-op.
2. Each string has a corresponding transfer to the procedure which will handle that request.
3. The procedure checks the argument to make sure it is within range, and transfers. No stack frame is made for debug_data.
4. Storage is provided for status variables which must be communicated between procedures.

If a new request is to be added to the probe package, the following must be done:

1. Add the request name to the end of table 1.
2. Add the appropriate transfer to the end of the transfer vector

dispatch_request makes the call

```
          call parse_scan$prime (line);
```

to place the input request line into parse_scan's internal static storage. When parse_scan$(atom or segname) is called, the line is scanned character by character. When a break character is found, the scan is stopped and the string delimited by the break character is returned as the first argument. The terminating break character is returned as the second argument.

On the next call to parse_scan the scanning is resumed at the character just beyond the previous break.

If the scan is terminated by the end of the string, the last argument is set to "1"b and the second argument is meaningless.