DEC	title: SPL COMMAND LANGUAGE AND DEBUGGING SYSTEM	prefix/class-number.revision: SPLDS/W-17
unchecked / checked: Roger Sturgeon	authors: M. Greenberg, Roger Sturgeon	approval date: 9/10/69 — revision date:
approved: Mel		classification: Working Paper
		distribution: Company Private — pages: 11

ABSTRACT and CONTENTS

The debugging language for the M1 Compiler System is described in detail sufficient for the user. Informal syntax and semantics are given for all commands, including a careful treatment of possible program states with respect to suspended tasks.

The command processor will exist at any time in one of several modes. These are:

executive	&
verbose editor	*
quick editor	:
expert debugger	-
beginner debugger	=

Each of these modes will have its own herald as indicated. SPL is entered in executive mode and will thus type an &.

To change modes merely type the herald for the desired mode immediately after another herald and continue. This new mode will stick until another herald is typed.

The executive and editor modes will be discussed elsewhere. The debugger is now described.

A command in beginner mode is of the general form

=<command> [<modifiers>:] [arguments]

By <command> is meant a sufficient number of characters to distinguish the command from all others, terminated by a blank. The modifiers are a collection of single letters terminated by a colon. The nature of the arguments depend on the command. The entire command is terminated by a carriage return.

A command in expert mode is of the general form

-[<modifiers>] <command> [<args>]

The command will be a single punctuation character and will thus serve to terminate the modifiers and eliminates the need to type a blank after the command.

Command descriptors

MODE #

This command takes no arguments (other than modifiers) and sets the debugger permanently into all the specified modes.

GOTO $

Takes two optional arguments. First is the editor address of the statements to transfer control to. Second is number of break points to pass through before control is returned to the debugger.

CONTINUE ,

Takes one optional argument, the number of break points to pass through. This command will ignore a break on the first statement executed.

STEP +

Takes one optional argument, the number of statements to execute.

BREAK .

Takes one argument, which is an editor address or interval at which to set a breakpoint(s).

KILL [

Takes one argument, an editor address or interval of breakpoint(s) to clear.

DISPLAY]

No arguments, lists all breakpoints.

TRACE ;

Takes two arguments, the first is an editor address. The trace breakpoint is set to this address and the program 'continues.' The second argument specifies the number of breakpoints to pass through.

REPORT (

The user may specify that the value of any number of expressions be printed out at a break. Executing the REPORT command (takes no arguments) puts the debugger in a state where it is editing a line interpreted as a list of expressions separated by commas, whose values should be printed in the break message.

NOTES:

Whenever a program breaks, a break message is printed. The message will always start with the break address. The statement where the break occurred becomes the "current line" for the editor. To cause the source statement where the break occurred to be printed, use the L modifier with the program execution commands (CONTINUE, STEP, TRACE, GOTO). To suppress this, use the Q ("quiet") modifier.

A conditional breaking facility can be evoked in the following way: If at the time control is transfered to the user program, a function with name BREAK' is defined, then the debugger will cause the program to be interpreted and the function BREAK' will be called after every statement is executed. Thus any user specified condition can be tested for after every statement.

To clear all break points use the modifier A with the KILL command.

To set or clear the trace break point use the modifier T with the BREAK or KILL command.

To cause a break message at every breakpoint passed through use an E modifier with the STEP, CONTINUE, TRACE, and GOTO commands. To cause a break message only at the end use an N modifier. If no modifier is used, then the current mode will take effect.

The user can specify that the GOTO, TRACE and CONTINUE commands only count the breakpoint on the current statement when counting the number of breakpoints passed through, by using the H modifier. The G modifier causes any breakpoint encountered to be counted.

Notes on transfers of control to user program: The debugger has the facility for maintaining two programs at once. They

will hereafter be referred to as Task 1 and Task 2. The debugging may exist in a state called the zapped state (reset state) in which the call stack and hardware stack are initialized and the debugger is not aware of any tasks. Using a Z modifier with the mode command will zap the state. When a transfer of control statement is executed a number of things must be considered before transfer of control is allowed:

1) Do any statements or functions need recompiling? If so, they are recompiled. If the break statement has moved the P-counter must be changed accordingly.
2) Is continuing legal? It isn't if the break statement has been modified or if the call stack cannot be unwound correctly (see below).
3) Under what task should the program be run?
4) Should the call stack be unwound and how much?

Control can be transfered to the program by the GOTO, CONTINUE, TRACE and STEP commands or by a direct statement. CONTINUE, TRACE and STEP are legal only if

a) a Task 1 program exists
b) the P-counter is not in the middle of the statement Task 1 broke in, if the statement has been modified since the break.

To transfer control, first all modified statements or functions are recompiled. Then the action taken is determined by the following diagram.

	call stack not empty	call stack empty
GOTO command or direct statement containing GOTO or RETURN	unwind stack transfer control as Task 1	illegal
otherwise	unwind stack transfer control as Task 2	transfer control as Task 1 with call stack set to dummy local environment

Unwind stack: The stack must be unwound far enough so that any call on the stack, in a statement that was subsequently modified, is removed from the stack. Once this is accomplished the stack must be further unwound until the top function on the stack is the same as the current editor function. Unwinding the call stack requires user confirmation. If the stack is unwound the P-counter is set to the instruction after the top call on the unwound stack.

Run as Task 2: This means the state of Task 1 and the hardware stack pointer are saved. When Task 2 terminates for any reason the saved state and pointer are restored.

EXAMINE /

This command takes Ø, 1, or 2 arguments.

If Ø, then the last quantity printed is printed again

If 1, then the value of the expession is printed

If 2, then the arguments are interpreted as bounds and everything in between is printed inclusive.

The value of an expression can be printed in one of the following formats

signed integer	I
unsigned integer	U
real number	R
6-bit characters	6
8-bit characters	8
pointers (with B, S, I, U)	P
double precision (with R or 0)	D
complex	X
unsigned octal	O
field descriptor	F
label or function	B
string	S
machine code	M
longlong	W

The type of the root operand in the parse tree will determine the format in which the value of an expression will be printed

unless a permanent mode is set. To suppress the permanent mode use a V modifier.

Another format may be specified instead by using one of the above modifiers. The examine command will abort if what is requested doesn't make sense.

For the two argument variety of the command both arguments must be references (in the sense of the SPL manual) i.e., be simple or have a principal operator which is indirection or subscripting, to be meaningful. Further, both arguments must specify addresses in the same environment, i.e., the same common block or function, if they are both simple variables or subscripted variables.

The two arguments otherwise are interpreted as absolute addresses and the contents of the cells between the two addresses are printed in the specified mode.

NEXT >

This command takes one optional argument, n. It prints the next n quantities following the last quantity (location) printed. If the argument is missing, it is taken to be 1. The format used for each quantity printed is taken from the symbol table unless overridden by modifiers in the command. Reaching the end of an environment (function, common block, absolute) terminates the command.

PREVIOUS <

This command works the same as NEXT except that the n preceding, rather than following, quantities are printed.

LEVEL ↑

This command takes two arguments. The first specifies the number of levels of local environment on the stack to jump back. A negative number means go forward on the stack. The second argument if specified is a function name. In this case jump n levels of that function.

FIND)

Not yet specified.

All state registers, etc., will be put in fixed places in the current global environment by the debugger and may be referred to with built-in symbols, to wit (in order):

PC'	program counter
AR', BR', CR', DR'	4-word accumulator
XR'	index register
LR'	local environment
GR'	global environment
SR'	status register

block name

TABLE OF MODES

Mode	Used With	Meaning
A	KILL	clear all breakpoints
B	EXAMINE, MODE	label or function
C		
D	EXAMINE, MODE	double precision (R,O)
E	GOTO, CONTINUE, STEP, TRACE, MODE	print message at every breakpoint passed
F	EXAMINE, MODE	field descriptor
G	GOTO, CONTINUE, TRACE, MODE	all breakpoints encountered are counted
H	GOTO, CONTINUE, TRACE, MODE	only breakpoint on current statement is counted
I	EXAMINE, MODE	signed integer
J		
K		
L	GOTO, CONTINUE, STEP, TRACE, MODE	print source at break
M	EXAMINE, MODE	machine code
N	GOTO, CONTINUE, STEP, TRACE, MODE	print message only at end
O	EXAMINE, MODE	unsigned octal
P	EXAMINE, MODE	pointers (B,S,I,U)
Q	GOTO, CONTINUE, STEP, TRACE, MODE	do not print source at break
R	EXAMINE, MODE	real number
S	EXAMINE, MODE	string
T	BREAK, KILL	set/clear trace breakpoint

U	EXAMINE, MODE	unsigned integer
V	EXAMINE, MODE	suppress permanent mode
W	EXAMINE, MODE	longlong
X	EXAMINE, MODE	complex
Y		
Z	MODE	zap state
6	EXAMINE, MODE	6-bit characters
8	EXAMINE, MODE	8-bit characters