Inter-Office Memorandum

To	Mesa Group	Date	October 27, 1977
From	Barbara Koalkin	Location	Palo Alto
Subject	Debugger Interpreter - Part 2	Organization	SDD/SD

XEROX

Filed on: [MAXC]<KOALKIN>DWG1.bravo

DRAFT

This memo represents a summary of today's discussion on the proposal for the debugger interpreter as well as other topics of interest regarding the Mesa 4.0 debugger.

The basic thought seems to make the interpreter simple but complete. The grammar for the interpreter may be found on the attached pages. It was decided to make the syntax itself very general in order to allow room for expanding, however we intend to limit what will be included in the first implementaion.

It was decided to "buy" the front end of the compiler and modify it to fit the grammar of the interpreter. We will need to modify the scanner, modify the parser to accept the reduction rules of the debugger, and generate a module to do the actual interpreting.

The question of context brought up some interesting discussion. It was decided that values will be interpreted in terms of the current context (by default). However, you may specify a particular module or definitions file by typing name$foo (more on this syntax later).

On the issue of what kind of user interface the interpreter should have, it was decided to keep the present set of INTERPRET commands and have the new interpreter run off a separate command (SP was proposed) until its reliability is well proven.

It was decided to do a two pass interpreter; parse once looking for valid syntax, if everthing looks ok then do the second parse for evaluating on the fly. This method was chosen so as to avoid the canonical case of having imbedded procedures calls causing core swaps (with high time costs) before finding a simple syntax error.

The QualifiedAccess question reduced to the following two cases:
ie., you want to have p.q
(1) if p is a pointer to a record, use the "."
(2) if p is a module name or definitions file, use the "$"
The following lookup algorithm was decided on:
$ means look through the current context and its imported contexts, followed by the module table of the BCD and then look for a file. If the time cost proves to be too high using this algorithm we will consider introducing a new notation to distinguish between the case of naming a defs file from an included context vs. naming a random file on the disk.

To accomplish our goals some new notation was proposed:

Interval will be a new type of qualification. The initial implementation will be only for printing purposes and therefore an Interval will not be allowed on a left side or embedded inside other expresssions. The syntax looks as follows:

LeftSide := . . . | (Expression) Qualifier | LeftSide Qualifier | MEMORY [Interval]
Qualifier := . . . | [Interval]
Interval := Expression .. Expression | Expression ! Expression
which means either begin at the first expression and go until the second expression or begin at the first expression and go until the first expression + the second expression.

Various suggestions were propsed for simplifying the LOOPHOLE notation. It was decided to adopt % as a symbol instead of having to type LOOPHOLE (as Richard said, "Loop on top, hole on bottom"). The syntax looks as follows:

LeftSide := . . . | (Expression) Qualifier | LeftSide Qualifier
Qualifier := . . . | % TypeSpecification | %

@TYPE means POINTER TO TYPE - ie., 133B % @Foo↑ means LOOPHOLE 133B to be a POINTER to a Foo and show me what that Foo looks like

@variable will either mean POINTER TO its TYPE or POINTER TO UNSPECIFIED, it is unclear at this time

Multiple expression evaluation will be allowed. The syntax looks as follows:

StmtList := Stmt | StmtList ; Stmt

The following topics involving the interpreter were discussed, although it was decided to punt on most of these issues for now:

Procedure calls will be allowed of the same flavor that is currently allowed; ie., you can not interpet call any nested procedures or procedures with more than 5 parameters

The capability of having user defined temporary variables sems useful, however it was decided to save this for possible future implementation.

Further qualified access of the form Config.module.var - ie. Mesa>Strings.foo would be useful, although too ambitious for now. This would help simplify the context switching mechanism for configurations that we currently have.

In talking about putting the interpreter only into the external debugger, discussion of the external/internal debugger split was brought up. This is a separate issue from the design of the interpreter and requires lots of thought on its own although it would be very useful in freeing up some space in the debugger for the interpreter.

It was decided to punt on doing string assignments for now, since there seems to be many complications.

It was decided to keep Array Type Constructors and Keyword Components out of the grammar at this time but they appear to be candidates for later inclusion.

The following topics regarding other debugger issues were discussed as well:

Conditional breakpoints (ie. "canned" program) might be helped by the interpreter but remain a separate issue to be discussed at some other time.

It would be nice to be able to RESUME signal with values as well as to pass parameters to NEW and START but this is not a high priority item.

It was agreed that caching variables will certainly speed up look-up time (as in a ← b; a), however this was considered to be an optimization and not required right now. The cache should contain an indication of what the value of the variable is and where to find it. Various suggestions were made regarding hash tables and other possibilities for organizing the tables.

The question of processes was also brought up. There needs to be a way of setting the context to a particular process, ie., a Set Process Context command which works with a process handle or PSB (process state block).

It was decided to add a comment command to the debugger command language. It was decided to use "--"; the semantics of which will mean to ignore the following characters until you see a CR. This will be useful for saving your own notes along with your typescript file.

Distribution:
Mesa Group

```
StmtList        ::= Stmt | StmtList; Stmt
AddingOp        ::= + | -
BuiltinCall     ::= LENGTH [ LeftSide ] | BASE [ LeftSide ] |
                    DESCRIPTOR [ Expression ] |
                    DESCRIPTOR [ Expression , Expression ] |
                    TypeOp [ TypeSpecification ]
Expression      ::= Sum
ExpressionList  ::= Expression | ExpressionList, Expression
Factor          ::= - Primary | Primary
Interval        ::= Expression .. Expression | Expression ! Expression
LeftSide        ::= identifier | ( Expression) Qualifier | LeftSide Qualifier |
                    MEMORY [ Interval ] | MEMORY [ Expression ] |
                    REGISTER [ Expression ] |
                    identifier $ identifier
Literal         ::= numericLiteral |        -- all defined outside the grammar
                    stringLiteral |
                    characterLiteral
MultiplyingOp   ::= * | / | MOD
Primary         ::= LeftSide | Literal | ( Expression ) | @ LeftSide | BuiltinCall
Product         ::= Factor | Product MultiplyingOp Factor
Qualifier       ::= . identifier | ↑ | % | % TypeSpecification |
                    [ ExpressionList ] | [ Interval ]
Stmt            ::= LeftSide | LeftSide ← Expression
Sum             ::= Product | Sum AddingOp Product
TypeConstructor ::= @ TypeSpecification
TypeIdentifier  ::= INTEGER | BOOLEAN | CARDINAL | CHARACTER |
                    STRING | UNSPECIFIED | identifier |
                    identifier $ identifier | identifier TypeIdentifier
TypeSpecification ::= TypeIdentifier | TypeConstructor
TypeOp          ::= SIZE
```

These are candidates for addition to the grammar. They appear to be desirable but not essential for the first implementation:

```
ArrayTC             ::= PackingOption ARRAY OF TypeSpecification
ComponentList       ::= KeywordComponentList
KeywordComponent    ::= identifier : Component
KeywordComponentList ::= KeywordComponent |
                         KeywordComponentList , KeywordComponent
PackingOption       ::= empty | PACKED
TypeConstructor     ::= ArrayTC
```