DECUS
PROGRAM LIBRARY

DECUS NO.	8-504A
TITLE	ESI (ENGINEERING AND SCIENTIFIC INTERPRETER)
AUTHOR	David J. Waks
COMPANY	Submitted by: Robert M. Supnik Applied Data Research Cambridge, Massachusetts
DATE	1966
SOURCE LANGUAGE	MACRO-8X

ESI (ENGINEERING AND SCIENTIFIC INTERPRETER)

DECUS Program Library Write-up DECUS NO. 8-504A

INTRODUCTION

ESI is an interactive programming system which greatly enhances the utility of the PDP-8, PDP-8/I or 8/S. It provides the small computer user with the computational language facilities normally found only on large systems. But more importantly, ESI is entirely self-contained. All tools for program creation, modification, execution, etc., are built into the system itself; no supporting systems are required. Because it concentrates these mechanisms in the form of a straightforward, "English"-like command language, ESI is ideal for the moderate-sized numerical problems that arise continually in engineering, science and system design and which must be solved by people who are not and may not wish to become programmers.

This document has two sections. The first is a guide through an initial session with ESI. The second is a short but rigorous exposition of the system. You are encouraged to experiment freely as you read Section I; ESI is self-protecting and it is unlikely that you can "blow" it inadvertently (or deliberately).

Underscored text such as

DELETE ALL. (input this)

is input to ESI which you are to transcribe to the

teletype. It is important to remember that all ESI statements begin with a verb and end with a period. The system will not "look" at an input line until carriage return is typed; carriage return will not be shown here. For recovery from typographical errors (you probably will make many at first), strike rubout to "erase" the most recently input character, or question mark-carriage return to nullify the entire line. If rubout is struck a number of times in succession, then an equal number of characters is "erased" from the right-hand end of the line.

The simplest level at which ESI can be used is that of a desk calculator. ESI will accept arithmetic expressions, evaluate them and return the result almost immediately. For example, input the following (remember to type carriage return):

TYPE 2 + 2.

Available arithmetic operators are:

+ addition
- subtraction
* multiplication
/ division
↑ exponentiation

In ESI-B (the 4K system) exponents must be decimal integers in the range 1 - 9. With ESI-X (8K), exponents may be arbitrary arithmetic expressions. Both systems depart slightly from conventional mathematical notation in that implicit multiplication is not recognized. Thus TYPE 2(7+3). is unintelligible to ESI (try it) and must be expressed as TYPE 2*(7+3). Use of parentheses is encouraged in ambiguous expressions like 1-2/B, which should be written as either (1-2)/B or 1-(2/B) according to intent.

Several often-used mathematical functions are built

into ESI-B. These are:

SQRT square root

SIN sine (argument interpreted as radians)

COS cosine (" " " ")

ABS absolute value

SGN signum (returns +1 or -1 according to sign of argument)

IP integer part

FP fraction part

In ESI-X, the list is extended to include:

LOG common logarithm

LN natural lorarithm

EXP exponential (e to a power)

ATN arctangent (returns angle in the range $-\pi/2$ to $+\pi/2$ radians)

DP digit part

XP exponent part

Function arguments must be enclosed in parentheses, and may be arbitrary arithmetic expressions. Thus

TYPE SIN(SQRT(5.17)/.283).

Invent a few examples yourself. When you feel comfortable with the keyboard and the notation, proceed.

If you have had experience with other programming systems you may already have seen that ESI does its

arithmetic in decimal.

TYPE 20*.1.

produces 2 rather than, say, 1.999999 because there is no error of conversion to and reconversion from an internal number base. Precise results therefore will often be preserved during computation. ESI's precision is, however, not infinite; normal roundoff errors will occur as with any finite precision computation tool. Internally, numbers and intermediate results are represented with a precision of seven decimal digits. Half-adjustment occurs at the eighth significant figure during both input and computation. Thus

TYPE 3.14159265.

and

TYPE 1/3, 2/3.

An important ESI construct is the FOR clause, used for iteration. Its general form is FOR<index>= a(b)c, <operation> in which a, b, and c may be arbitrary arithmetic expressions. The index is assigned an initial value of a. The operation is performed, b is added to the index and if c has been attained or exceeded in value by the index, the process ceases; otherwise, another iteration

is made. For example,

FOR I = 2(3)9, TYPE I.

or

FOR I = -3(-1)-7, TYPE I.

or even

FOR I = -SQRT(4)(2/4)6/2, TYPE I.

Using a variant of the TYPE statement, we can produce "useful" output:

FOR I = 0(1)5, TYPE "THE SQUARE OF "I" IS "I↑2.

Incidentally, when textual and numerical output are combined as above, ESI inserts no spaces; if you get numbers and text crowded together or too far apart, try again with appropriate spacing in the text string itself.

Another important element of the ESI language is the conditional- or IF clause. It has the general form IF <relation>, <operation>; the operation is performed conditionally, that is, if and only if the relation is true. For instance,

IF 10>2, TYPE "TEN IS GREATER THAN TWO".

The relational operators are:

< less than

LE less than or equal

= equal

GE greater than or equal

> greater than

NE not equal

The comparisons are of algebraic value. If magnitudes only are to be compared, the absolute value function must be used. As one would wish, both sides of a relation may be arbitrary arithmetic expressions. As another example of "useful" output we might execute

FOR I = 1(1)6, IF I↑7<3429, TYPE I.

which examines the positive integers 1 through 6 and types out those whose seventh power is less than 3429.

So far we have been using the direct or "desk calculator" mode of ESI in which statements are executed upon receipt and then discarded. In indirect mode ESI functions as a true computer operating on a stored program and its data.

Indirect statements, or steps, are required to have decimal numeric labels in the range 1 through 9.999999. Steps are filed in ESI memory according to a Dewey-decimal scheme so that to insert one between, say, steps 5.06 and 5.07 one need only tag the line with an intermediate number like 5.062. ESI construes an

input line as an indirect statement if and only if the line begins with a step number. Thus

TYPE "HELLO".

is direct and is executed at once, while

1.1 TYPE "HELLO".

is indirect and produces no apparent activity. That step 1.1 has in fact been stored is proven by

DO STEP 1.1.

or

TYPE STEP 1.1.

If we wish, we can modify a step by inputting a new line bearing the same number to be modified (try it with step 1.1). A given step can also be deleted, like this:

DELETE STEP 1.1.

Its absence can be confirmed by again inputting, say,

DO STEP 1.1.

ESI's response indicates that no such step exists presently.

The integer portion of a step number is termed a part number. A part is comprised of all steps having a given part number; it may be TYPEd, DELETEd and in particular, DOne as though it were a single step.

All but the most trivial computer programs require storage for intermediate computational results; in ESI, such storage registers are called variables and are designated by the letters A-Z. Values are stored into variables by means of the SET statement, which takes the general form SET <variable> = <arithmetic expression>. The arithmetic expression is evaluated and the resulting number stored in the indicated variable. What makes variables useful is that they may appear in arithmetic expressions as though they were numbers. Some examples (don't type them) of SET statements are:

```
SET Y = M*X+B.
SET T = SIN(A)/COS(A).
SET X = (-B+SQRT(B↑2-4*A*C))/(2*A).
```

The reason these examples should not be input is that ESI will not evaluate an arithmetic expression containing an undefined variable, i.e., a variable which has not had a value stored into it. For instance, Q presently has no value, so that

```
TYPE Q+Z.
```

produces an error message.

In addition to behaving as holders of single numbers as above, ESI variables can also act as arrays of no more than two dimensions. Subscripts can be arbitrary

arithmetic expressions, incorporating even subscripted variables, but must, after truncation to integral value, lie in the range -999 through +999. Subscript expressions are delimited by left and right square bracket, which are produced at the teletype by skift/K and shift/M respectively. ESI does not use array size declarations; instead, the dimensionality of a variable is indicated by the first executed SET statement storing into that variable. The array elements themselves are created as values are to be stored into them. Thus

FOR I = 1(1)4, FOR J = I(1)4, SET B[I,J] = Ø.

both establishes B as a two-dimensional array, and creates the upper triangular elements of the 4 x 4 matrix [B]. That only these elements have been created is shown by

TYPE ALL VALUES.

We are now ready to write a first ESI program -- a square root extractor. An equivalent program is built into ESI itself, but using ESI to create one will be instructive in learning the system and for seeing the value of "conversational" programming. Our program will require one input (the argument) and will produce one output (the square root) for which we shall use

the variables A and X respectively. Before thinking at all about how actually to take square roots, we can write a "supervisory" part to handle the clerical chores of input/output:

1.1 DEMAND A.

1.2 DO PART 5.

1.3 TYPE "THE SQUARE ROOT OF "A" IS "X.

1.4 TO STEP 1.1.

When executed, step 1.1 will cause ESI to request that you type in a value for A. Step 1.2 will then invoke the root-finding procedure, which will leave its result in X. Step 1.3 will output both A and X. Finally, step 1.4 will shunt "control" back to step 1.1 so that another square root can be taken without our having explicitly to restart the program.

We are now left with the problem of writing part 5. Since square root is a transcendental function we must employ an iterative technique, say, Newton-Raphson. The iterative formula for this is

$$X_{i+1} = 1/2(X_i + \frac{A}{X_i})$$

in which A is the argument, X_i is one approximation to the root and X_{i+1} is the next. We shall use A itself

as an initial approximation. X_i will go to $\sqrt{A}$ so that in the limit, $X_{i+1} = X_i = \sqrt{A}$; the process can be terminated when successive approximations produce no change in X_i. The code embodying these thoughts is shown below:

5.1 SET Y = A.

5.2 SET X = (Y+A/Y)/2.

5.3 IF X = Y, END.

5.4 SET Y = X.

5.5 TO STEP 5.2.

A, Y, and X respectively correspond to A, X_i and X_{i+1}. The operation END in step 5.3 causes control to leave part 5 and return to part 1.

To run the program, DO PART 1.; take the square root of several numbers and ascertain that the algorithm in fact works. When you have tired of this, strike altmode once to suspend execution (resumption can be effected by typing GO.) and once again to cancel the run.

We can rather easily modify the program so that it extracts cube roots. The iterative formula is

$$X_{i+1} = 1/3(2X_i + \frac{A}{X_i 2})$$

Appropriate modifications to the program are:

1.3 TYPE "THE CUBE ROOT OF "A" IS "X.

5.2 SET X = (2*Y+A/(Y↑2))/3.

Try taking the cube roots of 8, 27 and 81. Now try .999999 (six nines) and when you feel sure that something is wrong, cancel the run.

The program has been "looping"; and yet, if we examine X (type it out) we see that its value is quite close to $\sqrt[3]{.999999}$. One way to get a feel for the problem is to monitor the computation by typing each approximation as it is generated. This we can do by inputting

5.25 TYPE X.

Now restart the program with the same argument; when you have detected a pattern in the output, cancel the run.

As you see, the program has been repetitively generating two particular approximations to the root. The condition of equality of successive approximations is therefore not met; however, note that each number agrees with the actual root to within a few parts (five ought to be safe generally) in the seventh significant figure. Incorporating this convergence criterion into the program,

5.3 IF ABS((Y-X)/X)<5E-7, END.

The program now will never "hang up", and will compute all cube roots with a precision of ±5 in the seventh significant figure.

You might enjoy generalizing the program so that it takes n^{th} roots. The iterative formula is

$$X_{i+1} = \frac{1}{n}([n-1]X_i + \frac{a}{X^{n-1}})$$

If you are working with ESI-B, remember that an expression like N-1 is not a legal exponent.

PERMISSABLE FORMS IN ESI-B (VERSION 3 - 11/3/67):

DIRECT OR INDIRECT:

```
SET C = A*B+C*D.
SET C[I,J] = B[I-1,JT2]-C[I+1,J/2].
SET Y = IP(X/I).
FOR I = 1(1)N, SET A[I] = B[I]*C[I].

DO PART 3.
FOR R = 0(0.1)1.5, DO PART 2.
DO STEP 3.7.
FOR J = N(-1)1, DO STEP 7.352.
FOR I = 1.1(.1)1.9, DO STEP I+1.

TYPE 2+3*5.
TYPE X.
TYPE X, IP(X), SGN(X), ABS(X).
FOR I = 1(1)N, TYPE A[I].
TYPE "THIS IS A STRING".
TYPE "THE SQUARE OF "X" IS "XT2.
TYPE "ROW "I", COL. "J", VALUE IS "A[I,J].
TYPE STEP 2.3.
TYPE PART 6.
TYPE ALL PARTS.
TYPE ALL VALUES.
TYPE ALL.

DELETE X.
DELETE A[1,3], B[I,J], C, D.
DELETE ALL VALUES.
FOR I = 1(1)N, DELETE A[I].

LINE.
NOLINE.
```

DIRECT ONLY:

```
DELETE STEP 1.1.
DELETE PART 2.
DELETE ALL PARTS.
DELETE ALL.

X = 3*ABS(A-B)
Y=Z*R
Z = 14
A[1]=1.3E-6
B[999]=A[-999]
A[-43] = 1.414E+32
A[2] = 3175*I
```

INDIRECT ONLY:

```
1.1 TO STEP 1.7.
1.634297 TO STEP 3+.1*I.
1.7 TO PART 4.

2.3 END.

4.1 STOP.

6.1 DEMAND X.
7.35 DEMAND A[I,J].
8.1 DEMAND A[45].
```

CONDITIONAL CLAUSES:

```
IF A = B,
IF ABS((N-O)/N) < 1E-6,
IF IP(X) GE IP(Y),
IF SGN(X) NE 1,
IF (A-B)/C LE D-XT2,
```

NUMBERS:

```
2
3.141593
.003
0.01
-3.7E5
4.36E-7
-3.273E+17
```

OPERATIONS:

```
+  -  *  /  ()  ↑
```

RELATIONS:

```
<  >  =  GE  LE  NE
```

FUNCTIONS:

IP(X)	INTEGER PART
FP(X)	FRACTION PART
SGN(X)	SIGN PART
ABS(X)	ABSOLUTE VALUE
SQRT(X)	SQUARE ROOT
SIN(X)	SINE
COS(X)	COSINE

INTERRUPTED OR STOPPED:

```
ANY "TYPE" STATEMENT.
GO.
CANCEL.
```

NOTES:

← IS TYPED AND THE BELL RINGS WHENEVER A USER TYPE-IN IS REQUESTED.
[AND] ARE USED TO DENOTE SUBSCRIPTS.
? TYPED AT THE END OF ANY LINE CAUSES IT TO BE DISREGARDED.
"RUBOUT" DELETES THE PRECEDING CHARACTER AND TYPES ← TO SO INDICATE.
STEP NUMBERS ARE IN THE RANGE 1 TO 9.999999.
VARIABLES ARE THE SINGLE LETTERS A THROUGH Z.
"ALT MODE" INTERRUPTS EXECUTION OF A PROGRAM AT THE COMPLETION OF THE CURRENT STEP; ON A "DEMAND", "ALT MODE" CANCELS EXECUTION.

ESI-B OPERATING INSTRUCTIONS

I. Loading ESI into Memory

ESI is ordinarily loaded once and then only has to be reloaded if the computer has been used for programs other than ESI. ESI is completely protected from the effects of "bugs" in user programs and cannot be accidentally destroyed. The following procedure is used to load ESI-B:

A. If the RIM loader is not already in memory (and it usually is), insert it into memory through the switches. See the detailed instructions for this in the PDP-8 or PDP-8/S User's Handbooks.

B. Use the RIM loader to load the BIN loader through the ASR-33; directions for this are also contained in the PDP-8 or PDP-8/S User's Handbooks.

C. Use the BIN loader to load the paper tape labelled "ESI-B-MAIN-BIN" with either the ASR-33 or the high speed reader, if available. The AC should contain all zeroes after loading; if it is non-zero, the tape must be loaded again.

II. Adding and Deleting the Optional Functions

The ESI system is supplied as three paper tapes called "MAIN," "FUNCTION ADD," and "FUNCTION DELETE." In order to use the system, the MAIN tape is always loaded first as described above. The MAIN tape does not include the SIN, COS, or SQRT functions. These functions are added to memory by loading the separate FUNCTION ADD tape, which also reduces the amount of available user core from 800 to 600 words. The functions can be deleted, and the 200 words of user core regained, by loading the FUNCTION DELETE tape. Loading of these two tapes can be done at any time after loading ESI MAIN; they are loaded in the same way as ESI MAIN, as described in Step C above. Unless ESI has been run after loading ESI MAIN, Steps A and B are unnecessary; if ESI has been run, it is necessary to go through Step B as well as Step C. (That is, ESI clobbers the BIN loader.)

III. Starting ESI

ESI is started at location 5400, and should be restarted there after any halts. Note that stopping ESI and restarting at location 5400 does not do any damage to a user program currently in memory; it does, however, restart ESI in "direct mode" waiting for a direct command from the user.

IV. Punching an ESI-language Program on Paper Tape

A. To prepare leader-trailer:

1. Push punch "OFF" button.

2. Turn switch on front of ASR-33 to "LOCAL."

3. Push punch "ON" button.

4. Push and hold down in this order: SHIFT, CONTROL, REPEAT, and @ (shift of "P").

5. Hold all four buttons down for about 5 seconds.

6. Release buttons in the reverse order of pushing them.

7. Push punch "OFF" button.

8. Turn switch in front of ASR-33 to "LINE."

B. To dump a program:

1. Produce leader trailer as in A.

2. With punch "OFF," type "TYPE ALL." (Or "TYPE ALL PARTS." or "TYPE ALL VALUES.") Do not type the carriage return.

3. Push punch "ON" button.

4. Now type the carriage return.

5. When type-out is complete (when ← is typed), push punch "OFF" button.

6. Again produce leader trailer as in A.

V. Reading a Previously-Punched ESI-language Program

A. Place paper tape in ASR-33 reader.

B. Make sure ESI is in direct mode (back arrow ← typed at beginning of current line).

C. Push reader switch to "START."

D. At end of read-in, push reader switch to "STOP FREE."

VI. What to Do if You Run Out of Memory

A single block of memory is used by ESI to store user-program steps and subscripted array elements, as well as work space for ESI itself in the course of execution of programs. The single error message "STORAGE" is used to describe all three possible ways of running out of memory:

A. Not enough room to store the step just typed.

B. Not enough room to store another element of a subscripted array.

C. No room left for ESI work space.

The single block of memory used by ESI is continuously optimised so that running out of memory for one purpose leaves none for any other. In this way, all of user storage (less work space) is available for step storage if no subscripted arrays are used, or for subscripted arrays if no steps are stored.

One problem caused by this sharing of memory is that it is possible for ESI to be unable to interpret a direct command to delete some unwanted or unnecessarily-long step in order to gain additional space. In the same way, running out of memory while entering steps can leave ESI unable to interpret a "TYPE ALL PARTS" command. A method is available to permit the user to circumvent this problem, at the expense of destroying any subscripted variable elements the user may have stored, and possibly also destroying the RIM loader, which is normally protected by ESI.

The method is very simple, and involves the manual modification of location 0063 in memory. After halting the machine, examine location 0063. If it is greater than 7730, set it to 7777 (this will result in destroying the RIM loader). If it is less than 7730, set it to 7750 (which will not destroy the RIM loader). Then restart ESI at location 5400; ESI will now have sufficient work space for the analysis of any TYPE or DELETE command. No attempt should be made to enter, type out, or modify subscripted variables until the next DELETE ALL or DELETE ALL VALUES is given; either of these commands will initialise location 0063 to 7750 which is its normal value when no subscripted variable elements are being used.

The user should bear in mind that while ESI pays no attention to the presence or absence of spaces in any context except text strings (enclosed in quotes), spaces do occupy space in user storage which might be better employed in storing more user statements. Thus the user is encouraged to use spaces liberally to improve the readability of short programs, but is advised to discard readability for feasibility if space is going to be a problem.

PERMISSABLE FORMS IN ESI-B*

DIRECT OR INDIRECT:

```
SET C = A*B+C*D
SET C [ I,J ] = B [ I-1,J ↑ 2 ] -C [ I+1,J/2 ]
SET Y = IP(X/I).
FOR I = 1(1)N, SET A [ I ] = B [ I ] *C [ I ].

DO PART 3.
FOR R = Ø(Ø.1)1.5, DO PART 2.
DO STEP 3.7.
FOR J = N(-1)1, DO STEP 7.352.


TYPE 2+3+5.
TYPE X.
TYPE X, IP(X), SGN(X), ABS(X).
FOR I = 1(1)N, TYPE A [ I ] .
TYPE "THIS IS A STRING".
TYPE "THE SQUARE OF " X "IS" X ↑ 2.
TYPE STEP 2.3.
TYPE PART 6.
TYPE ALL PARTS.
TYPE ALL VALUES.
TYPE ALL.

DELETE X.
DELETE A [ 1,3 ] ,.B [ I,J ] , C, D.
DELETE ALL VALUES.
FOR I = 1(1)N, DELETE A [ I ] .

LINE.
```

DIRECT ONLY:

```
DELETE STEP 1.1.
DELETE PART 2.
DELETE ALL PARTS.
DELETE ALL.

X = 3*ABS(A-B)
Y=Z+B
Z = 14
A [ 1 ] =1.3E-6

A [ 43 ] = 1.414E+32
A [ 2 ] =3175*I
```

INDIRECT ONLY:

```
1.1 TO STEP 1.7.
1.7 TO PART 4.

2.3 END.

4.1 STOP.

6.1 DEMAND X.
7.35 DEMAND A [ I,J ] .
8.1 DEMAND A [ 45 ] .
```

CONDITIONAL CLAUSES:

```
IF A = B,
IF ABS( (N-O)/N) < 1E-6,
IF IP(X) GE IP(Y),
IF SGN(X) NE 1,
IF (A-B)/C LE D-X ↑ 2.
```

NUMBERS:

```
2
3.141593
.ØØ3
Ø.Ø1
-3.7E5
4.36E-7
-3.273E+17
```

OPERATIONS:

```
+ - * / ( ) ↑
```

RELATIONS:

```
< > = GE LE NE
```

FUNCTIONS:

IP(X)	INTEGER PART
FP(X)	FRACTION PART
SGN(X)	SIGN PART
ABS(X)	ABSOLUTE VALUE

INTERRUPTED OR STOPPED:

```
ANY "TYPE" STATEMENT.
GO.
CANCEL.
```

← IS TYPED AND THE BELL RINGS WHENEVER A USER TYPE-IN IS REQUESTED.
[AND] ARE USED TO DENOTE SUBSCRIPTS.
? TYPED AT THE END OF ANY LINE CAUSES IT TO BE DISREGARDED.
"RUBOUT" DELETES THE PRECEDING CHARACTER AND TYPES ← TO SO INDICATE.
STEP NUMBERS ARE IN THE RANGE 1 TO 9.999999.
VARIABLES ARE THE SINGLE LETTERS A THROUGH Z.

ESI-B OPERATING INSTRUCTIONS -- PRELIMINARY

DATE: 11/3/66

I. LOADING ESI-B

1. LOAD RIM LOADER MANUALLY (SEE PDP-8 MANUAL).
2. LOAD BIN LOADER THROUGH RIM LOADER.
3. LOAD ESI-B WITH BIN LOADER.

II. LOADING A PROGRAM PUNCHED BY ESI-B.

1. PLACE PAPER TAPE IN ASR-33 READER.
2. MAKE SURE ESI IS IN DIRECT MODE (BACK ARROW ← TYPED AT BEGINNING OF CURRENT LINE).
3. PUSH READER SWITCH TO "START".
4. AT END OF READ-IN, PUSH READER SWITCH TO "STOP FREE".

III. DUMPING AN ESI-LANGUAGE PROGRAM.

A. TO PREPARE LEADER-TRAILER:

1. PUSH PUNCH "OFF" BUTTON.
2. TURN SWITCH ON FRONT OF ASR-33 TO "LOCAL".
3. PUSH PUNCH "ON" BUTTON.
4. PUSH AND HOLD DOWN IN THIS ORDER: SHIFT, CONTROL, REPEAT, AND @ (SHIFT OF "P").
5. HOLD ALL FOUR BUTTONS DOWN FOR ABOUT 5 SECONDS.
6. RELEASE BUTTONS IN THE REVERSE ORDER OF PUSHING THEM.
7. PUSH PUNCH "OFF" BUTTON.
8. TURN SWITCH IN FRONT OF ASR-33 TO "LINE".

B. TO DUMP A PROGRAM:

1. PRODUCE LEADER TRAILER AS IN A.
2. WITH PUNCH "OFF", TYPE "TYPE ALL." (OR "TYPE ALL PARTS." OR "TYPE ALL VALUES."). DO NOT TYPE THE CARRIAGE RETURN.
3. PUSH PUNCH "ON" BUTTON.
4. NOW TYPE THE CARRIAGE RETURN.
5. WHEN TYPE-OUT IS COMPLETE (WHEN ← IS TYPED), PUSH PUNCH "OFF" BUTTON.
6. AGAIN PRODUCE LEADER TRAILER AS IN A.