DOCUMENT RESUME

ED 060 628 EM 009 634

AUTHOR	Bork, Alfred M.; Mosmann, Charles
TITLE	Teaching Conversations with the XDS Sigma 7. Systems Description.
INSTITUTION	California Univ., Irvine. Physics Computer Development Project.
SPONS AGENCY	National Science Foundation, Washington, D.C.
PUB DATE	26 May 71
NOTE	25p.
EDRS PRICE	MF-$0.65 HC-$3.29
DESCRIPTORS	Computational Linguistics; *Computer Assisted Instruction; *Computer Programs; Program Descriptions; Programed Instruction; Programing; *Programing Languages; Programing Problems
IDENTIFIERS	Macros; Metasymbol; Sigma 7

ABSTRACT

Some computers permit conventional programing languages to be extended by the use of macro-instructions, a sophisticated programing tool which is especially useful in writing instructional dialogs. Macro-instructions (or "macro's") are complex commands defined in terms of the machine language or other macro-instructions. Like terms in higher-order languages they can expand to a variable number of actual machine instructions. The system described here is based on the use of the macro-assembler of the Sigma-7 computer, called Metasymbol. Metasymbol allows for the use of machine language, the definition and use of macro-instructions, and the inclusion of FORTRAN subroutines. This system allows the teacher considerable flexibility in composing instructional dialogs. Specifics of programing are discussed, and an example computer run given. (RB)

pcdp

TEACHING CONVERSATIONS WITH THE XDS SIGMA 7

System Description

Alfred M. Bork
Charles Mosmann

University of California
Irvine, California 92664

May 26, 1971

physics computer development project, university of california, irvine, 92664

CONTENTS

SIGMA 7

CONTENTS

rsity of california, irvine, 92664

INTRODUCTION

Computer dialogues for instructional purposes are sometimes written in specialized languages developed for just that purpose (Coursewriter, Foil, Planit). Such languages have special advantages, particularly for authors unskilled in computer programming; but limitations may exist and the skilled programmer is prevented from using the full power of the computer. Also, they are available only on particular machines. An alternative is the use of general purpose computer languages; such languages as SNOBOL, PL/1, JOSS, BASIC, and FORTRAN have been used for teaching conversations. The advantage of such languages is that most of the facilities of the computer are available to the teacher, although practically it may be difficult to employ particular facilities. Further, they are widely available. A third alternative--the use of machine languages--has almost never been considered. Although it allows maximum flexibility to the programmer, coding directly in machine language tends to be slow, complex, and expensive. However, some modern assembly programs permit machine language to be extended by the use of macro-instructions. Using such a system, the programmer has the facility to extend the language by adding his own complex commands (called "macro-instructions" or "macros") and defining them in terms of the machine language or other macros. Like terms in higher-order languages, these macro-instructions, can expand to a variable number of actual machine instructions, thus providing sophisticated programming tools.

al purposes are sometimes written
for just that purpose (Course-
ages have special advantages,
in computer programming; but
led programmer is prevented from
er. Also, they are available
lternative is the use of general
anguages as SNOBOL, PL/1, JOSS,
for teaching conversations. The
t most of the facilities of the
her, although practically it may
facilities. Further, they are
tive--the use of machine lan-
idered. Although it allows
mer, coding directly in machine
, and expensive. However, some
chine language to be extended
Using such a system, the pro-
the language by adding his own
structions" or "macros") and
ine language or other macros.
es, these macro-instructions, can
ual machine instructions, thus
g tools.

The system described here is based on the use of the macro-assembler of the Sigma-7 computer, called "Metasymbol." Metasymbol allows for the use of machine language, the definition and use of macro-instructions, and the inclusion of FORTRAN subroutines. Thus the user of Metasymbol has the total flexibility of machine language with both FORTRAN and added facility to add the terms he needs for his particular application.

These capabilities make a macro-assembler extremely attractive for use in developing a system for producing instructional dialogues. In using it as the basis for our system, we cannot say that we have developed a <u>language</u> for composing conversational programs; nor have we written a <u>compiler</u> or <u>interpreter</u> of some language. The system described here is simply a flexible set of macro-instructions which, added to the existing capabilities of Metasymbol and the BTM time-sharing monitor on the Sigma-7, allows a teacher considerable flexibility and convenience in writing instructional dialogues. The system is now available and is being used to write instructional dialogues. It should be clear that such a system is never "complete"--one of its major advantages is the ease with which additional features can be added and new terms be defined. The system depends, obviously, on the facilities of Metasymbol and the Sigma-7; but the idea could easily be adapted to any computing system with a macro-assembler of the general type of Metasymbol.

A SIMPLE OVERVIEW

An example will help to introduce the subject. The following flow chart illustrates a small part of an instructional dialogue. The situation is trivial, presumably for a first grader; it is used here only for illustrative purposes.

A message is typed to the student; a student response, typed at the terminal, is accepted; various scans are made on the input information and, depending on that input, decisions are made and responses are typed to the student. The rounded boxes on the chart contain messages to be typed and the square box shows the responses anticipated. In this example, first the message "What is 5 x 6?" is typed. If the student responds with the correct answer, we tell him, "good"; if he is adding, we tell him and let him try again. In all other cases, he is simply asked to try again.

ect. The following flo
uctional dialogue. The
st grader; it is used

ent response, typed at the
made on the input informa-
ons are made and responses
xes on the chart contain
hows the responses antici-
e "What is 5 x 6?" is typed.
answer, we tell him, "good";
try again. In all other

The corresponding program segment is shown below. The WRITE statement leads to the message appearing on the student terminal. INPUT types a question mark and waits for student input. IF checks to see whether certain strings are present in what the student types and branches if so. OTHER shows where control is to go if one of the anticipated inputs is not found. Some of the lines begin with labels which allow the IF and TO statements to refer to these points in the program.

```
            SYSTEM DIALOG
            . . .
MES1        WRITE     'WHAT IS 5 x 6?'
            INPUT
            IF        '11',MES2
            IF        'ELEVEN',MES2
            IF        '30',MES3
            IF        'THIRTY',MES3
            OTHER     MES4
MES2        WRITE     'YOU ARE ADDING. TRY AGAIN'
            TO        MES1
MES4        WRITE     'NO. TRY AGAIN'
            TO        MES1
MES3        -----
            -----
            . . .
            END DIALOG
```

DESCRIPTION OF COMMANDS

The simple example shown above used only a few commands; a large repertory is available. The following paragraphs offer brief descriptions of most of them, grouped more or less by function. Some more esoteric ones are omitted for the sake of brevity; there are also a number of synonyms of the more important commands which are not included here. The intention is to give the reader a feeling for the kinds of capabilities available. The Users Manual includes precise and more detailed descriptions of the functions of each of the commands.

1. Formal Statements

It is important to remember that the system is actually no more than a set of terms or commands which are defined so that they may be used with Metasymbol. Thus the conventions of that program (and the operating system in which it is embedded) must be adhered to. The conventions for format of statements, legal statement labels, and so on, are described briefly below in Chapter 4 ("Grammatical Rules") and will be found in detail in documentation of Sigma-7 software. (See Bibliography.)

> SYSTEM. The first statement in a program must be SYSTEM DIALOG. This informs the Metasymbol assembly program that the program which you have written uses the dialogue macro-instructions described here.
>
> NAME. The author may wish to assign a unique name to his instructional program. If he wants it to be called "PHYS," for instance, the second statement of his program should be NAME 'PHYS'.

few commands; a large
agraphs offer brief des-
less by function. Some
e of brevity; there are
rtant commands which are
ve the reader a feeling
The Users Manual includes
the functions of each of

m is actually no more than
ed so that they may be used
that program (and the
must be adhered to. The
l statement labels, and so
r 4 ("Grammatical Rules")
ion of Sigma-7 software.

a program must be
e Metasymbol assembly
you have written uses
described here.

ssign a unique name to
he wants it to be called
d statement of his pro-

END. The last statement of any program must be END. The statement END XYZ indicates that this is the last statement of the program and also that the statement labelled XYZ is the first statement to be executed when a student uses the instructional dialogue. The statement END DIALOG indicates that the program is to begin with the NAME or START command.

2. Displaying information to the student

WRITE. This is the basic command for causing information to be typed (or otherwise displayed) at a student console. The argument (that part of the statement which follows the word WRITE) can be the information itself, enclosed in single quotes: WRITE 'THIS IS A STRING'. Or it can refer to a string by name rather than directly: WRITE MES8 causes the string of characters stored at MES8 to be typed. (Section 5 will explain how strings are stored.) This is more convenient than repeating the message if the same message is to be used in several places. Each execution of a WRITE command begins with a carriage return and line feed, starting a new line.

OUT. This command is the same as WRITE except that it will not begin a new line. This can be useful in making up a display line out of several strings which are stored or computed independently.

SKIP generates one or more blank lines. The argument indicates the number to be skipped; if the argument is omitted, one line is assumed. Thus SKIP 5 generates 5 blank lines; SKIP generates one.

GRAPH. The instruction GRAPH X,Y,20 will graph 20 points using numbers stored in X as the horizontal displacement and the corresponding values stored in Y as the vertical displacement. All numbers will be scaled.

3. Accepting information from the student

INPUT. This command will cause a carriage return, two line feeds and a question mark to be executed at the student terminal. Then it waits for the student to enter material. The student indicates that his message is complete by executing a carriage return. The maximum amount of material he can enter is set at 380 characters but this can easily be extended. (He can use line feeds for long inputs--he isn't expected to get 380 characters on one line!)

INBELL is an alternative to INPUT. It does not return the carriage or type a question mark; rather a bell is sounded to indicate that input is expected. This form is useful for completing equations or sentences or filling in tabular data.

4. Analyzing student input

IF is a flexible command for examing student input. Some examples may best display its power. IF 'VELOCITY',T34 means that if the word VELOCITY is anywhere in the material which the student has just typed, then the next statement to be executed is that labelled T34. IF M3,T34 means if the string at M3 is in the input, go to T34. IF ('HORSE','COW', 'PIG'),T34 means that the branch is to occur if any of the three strings appears. If the conditions of an IF statement are not met, the next statement in sequence is executed.

IFONLY. The IF command searches for a match anywhere in the student input; IFONLY calls for an exact match between the indicated string and the student input.

IFNULL. Students often enter nothing, pushing only the carriage return key. This command checks for this condition and branches to an author-supplied message.

IFYES checks for various possible affirmative replies.

IFBEFORE takes into account the relative position of symbols in the response. It refers to the last successful match in an IF statement. For example, the sequence:

```
        IF      'ENERGY',E1

        ...

E1      IFBEFORE      'POTENTIAL',E2
```

tests first to see whether the word ENERGY appears anywhere in the input and, if it does, then tests

5. Manipu

Before att

the studen

The string

he can enter is set at
n easily be extended. (He
g inputs--he isn't expected
ne line!)

INPUT. It does not return
tion mark; rather a bell is
put is expected. This form
uations or sentences or

r examing student input.
lay its power. IF
if the word VELOCITY is
hich the student has just
ment to be executed is
T34 means if the string
to T34. IF ('HORSE','COW',
branch is to occur if any
rs. If the conditions of
t, the next statement in

arches for a match any-
IFONLY calls for an
dicated string and the

ter nothing, pushing only
This command checks for
s to an author-supplied

ssible affirmative

t the relative position
. It refers to the last
statement. For example,

TIAL',E2

the word ENERGY appears
if it does, then tests

to see whether the word POTENTIAL appears in the string after the last matched word.

IFFILTH has been found useful in some contexts: it checks the input line for a number of common examples of objectionable language.

OTHER. A series of IF statements may be followed by an OTHER which indicates the action to be taken if none of the conditions specified in the IF statements occur. It converts to a GOTO or unconditional branch instruction.

5. Manipulating Strings

Before attempting to match a string, it may be desirable to modify the student input, putting it into some more standarized format. The string manipulation commands provide this capability.

NOBLANK. One problem in matching formulae or equations is that blanks may appear in random places. This command removes all blanks from the input string.

DELETEALL removes from the input all occurances of a specified string. Thus, DELETEALL 'AND' removes the word "and" everywhere it appears in the student input.

SUBALL. The author may wish to modify the input in other ways. This command replaces all occurances of one symbol string with another. Thus, SUBALL'**',↑' replaces the double asterisk with an up-arrow whereever it appears.

ADDAST, which takes formula input by students and transforms it into a BASIC-like form. It inserts asterisks between letters, or between numbers and letters; it replaces the FORTRAN double asterisk with an up-arrow; it deletes blanks; etc.

MOVE. Sometimes it is convenient to store all or part of an input string for later use. The various forms of the MOVE command facilitate this. MOVE A,B moves the entire string A to location B. MOVE A,B,5 moves the first five characters from A to B.

DEFINE. To secure space for moving, it is necessary to define a label and indicate how long a string it is to hold. The statement DEFINE B,50 reserves space for fifty characters, with the label B.

STRING. If characters are to be stored into a string initially, this command is used. L3 STRING 'VELOCITY' indicates that the eight character string "VELOCITY" is to be stored at label L3. (Strings are stored in a fashion different than that used in most Sigma-7 software. The first word contains the number of characters; the characters begin in the second word.

6. Manipulating numbers

Numbers typed by students appear initially as characters in strings. To use these numbers in internal calculations, it is necessary to convert them to a suitable form. To display the results of computation the reverse conversion is needed. The following commands facilitate these conversions.

NUMBER. An input string which should be only a number in character form can be converted into a real number by means of this command. The statement NUMBER TIME, NOGOOD will examine the contents of the input, either converting it to floating point form and storing it at TIME or (if this is not possible because of extraneous characters, etc.) branching to NOGOOD.

SCAN. To separate a string into the part containing a number and the strings before and after the number, one can use this command. SCAN NUMST,STBEF,STAF,ERR means store the number string in NUMST, the characters preceding it in STBEF, and the characters following in STAF; if the string has no recognizable number, branch to ERR.

SCAN# performs the same functions as SCAN but converts the numerical portion into a floating point number, rather than transferring it as a character string (i.e., it is equivalent to SCAN followed by NUMBER).

IFNUMEX will test a string to see whether it is a number. The command branches to the specified location if the string is exclusively a number.

AROUND tests a floating point number (N) to see whether it is within a given range (E) of a number quantity (S) and if so, branch (GOTO): AROUND N,S,E,GOTO.

o be stored into a string
sed. L3 STRING 'VELOCITY'
racter string "VELOCITY" is
Strings are stored in a
used in most Sigma-7 software.
number of characters; the
nd word.

ally as characters in strings.

lations, it is necessary to

isplay the results of com-

eded. The following commands

ich should be only a number
nverted into a real number
The statement NUMBER TIME,
tents of the input, either
oint form and storing it at
sible because of extraneous
to NOGOOD.

into the part containing a
re and after the number, one
NUMST,STBEF,STAF,ERR means
NUMST, the characters
the characters following in
recognizable number, branch

ctions as SCAN but converts
a floating point number,
as a character string
SCAN followed by NUMBER).

to see whether it is a
es to the specified location
y a number.

int number (N) to see whether
(E) of a number quantity (S)
AROUND N,S,E,GOTO.

BETWEEN tests the size of a floating point number. BETWEEN N,BOTTOM,TOP,GOTO tests whether N is greater than BOTTOM and less than TOP and branches to GOTO if this is the case.

DEFNUM reserves storage for a number which is to be used in the program. The argument is the name (or names, if more than one) of the variable.

NUMWRITE starts a new line and then prints from one to four numbers.

RANDOM generates random numbers. RANDOM X,A,B stores in X a random number between A and B, or, if A and B are omitted, between 0 and 1.

7. Using Counters

Often in a dialog the instructor wants to note how many times the student has been through a given loop, whether he has taken one branch or another, or his past performance, to determine where the student should go next. The mechanism for doing is as though setting and testing counters.

COUNTER is the command to identify the name of counter and set its initial value. COUNTER A indicates that A is a counter with an initial value of zero. COUNTER B,5 indicates that B is a counter with the initial value of 5. COUNTER (C1,C2,C3) indicates that three counters are to be established with initial values of zero.

BUMP increases the value of a specified counter or counters by one: BUMP (C1,C2).

DECREASE decreases the value of the specified counter(s) by one.

RESET resets the specified counters. RESET (A,B),3 resets A and B to 3. If the number is omitted, zero is assumed.

ADDCOUNT adds the contents of specified counters: ADDCOUNT A,B adds counter A to counter B and stores the sum in A.

TO with a single argument is a simple unconditional branch statement. The statement TO FRAME4,(ZZ,GE,5), however, will cause a branch to FRAME4 if and only if counter ZZ is greater than or equal to 5. The second part of the second argument indicates the logical relationship and can be any of the following, with the usual meaning: GT, NE, LT, LE, EQ, GE. If it is omitted, GE is assumed.

SWITCH is an alternate command for testing a counter. It works like a FORTRAN computed GOTO. SWITCH A,(A0,A1, A2,A3,A4) indicates the labels of locations to be branched to when the counter A has the values of 0, 1, 2, 3, and 4; if A is greater than 4, the next statement is executed.

8. Restart

Sometimes it is inconvenient for a student to finish an entire conversational lesson in one sitting. A facility to let him pick up where he left off keeps him from going through the whole thing again.

ENTRY is the command which permits restart. Whenever one is executed as a student uses the program, its location and the student ID are saved on disk. Thus, the latest entry point which the student has reached is preserved. This is the point from which he will be restarted.

9. Saving Student Responses

The author may wish to save information about student responses for later study and as an input for later revisions of his program.

SAVE copies onto a disk file the current contents of the input buffer, togther with time and date. This record is identified by a name supplied as an argument to SAVE. The command can also be used to save the value of the counters.

SAVEID does the same thing but, in addition, saves the student's ID.

e unconditional
FRAME4,(ZZ,GE,5),
ME4 if and only if
to 5. The second
s the logical relation-
, with the usual
it is omitted, GE

esting a counter.
O. SWITCH A,(A0,A1,
ocations to be branched
s of 0, 1, 2, 3, and 4;
atement is executed.

finish an entire con-

y to let him pick up

h the whole thing again.

restart. Whenever one
rogram, its location
k. Thus, the latest
eached is preserved.
l be restarted.

student responses for

ons of his program.

urrent contents of
e and date. This
lied as an argument
sed to save the

addition, saves the

The teacher can access the files created by using available facilities of the Sigma-7 system, particularly "Ferret", or through special sorting programs.

10. Employing FORTRAN subroutines

It is possible to use the full power of FORTRAN within a dialog. This is particularly useful when a large amount of calculation is required, as for simulation.

> FORTRAN is the command to introduce such a subroutine. A typical command is FORTRAN POLLY,(X,Y,Z) where POLLY is the name of the FORTRAN subroutine and X, Y, and Z are the arguments for the subroutine. The routine itself must be compiled in the background using the FORTRAN IV compiler and loaded along with the rest of the program.

11. Ending the program

> STOP. At a point in the program where the student is to terminate because the lesson is over (this may not be the last statement in the program as the program is written), this statement is used to terminate the program.

GRAMMATICAL RULES

Only a few simple grammatical rules need to be followed in preparing programs. As indicated earlier, these are largely dictated by the conventions of Metasymbol and other systems programs on the Sigma-7.

A statement label can be placed in any statement but it is usually only useful if the statement is referred to by another statement. All labels must begin at the beginning of the line; an initial space indicates that the statement has no label. The same label can be used only once; multiple appearances would make references to that label ambiguous.

A command should be preceded and followed by at least one space. Some commands have several arguments: There should be no spaces between them as the first space after the arguments indicates the end of the statement. (Spaces within literals are an exception.)

Literal strings are enclosed in single quotes. To indicate a single quote as part of a string, two successive single quotes must be used: for example, 'THIS ISN''T CORRECT'.

to be followed in preparing
e largely dictated by the
ms programs on the Sigma-7.

atement but it is usually
to by another statement.
the line; an initial space
. The same label can be
d make references to that

by at least one space. Some
ould be no spaces between
ts indicates the end of the
n exception.)

otes. To indicate a single
single quotes must be used:

DEVELOPING AND IMPLEMENTING STUDENT-COMPUTER DIALOG

Developing and entering conversational computer teaching sequences can be done in many different ways. Each individual or group will evolve its own style for such work. This section suggests one possible procedure, attempting to involve people in the jobs they can perform most efficiently.

Design and coding

The initial task is the design of the material. This must involve people with both detailed knowledge of the subject matter and insight into effective ways of teaching it. These individuals may have little experience with computer programming and little desire to become involved with it. They will naturally require a good understanding of the kinds of things which are possible but they will want to produce the specifications for their dialogues in the form most convenient to them.

This might be a flow chart form--similar to the one shown above in Section Two. The material in this form could be independent of any particular computer system and of any language. The flow chart would serve other functions as well. A good flow chart is an excellent way to document a program. It will also be a convenient way for other teachers to look over the material quickly so as to decide whether they might want to use it with their students.

The next stage is to convert the flow chart into the statements which will make up the program and enter these statements into the computer. The system we have employed has been to type the statements directly from an on-line terminal using a text-editing program (EDIT) available on the Sigma-7. Since a vast amount of typing is necessary even for a simple dialogue, a very useful practice has been to use professional typists for the bulk of the work of transcribing programs from flow charts.

The typists who will do the work are shown some flow charts and given simple explanations of how they work. They are then introduced to the use of computer terminals and the conventions of the editing system. With some assistance at first, the typist is able to type directly from the flow chart. Thus, she knows that when she sees the square boxes, she will have to write a series of IF statements, and that the rounded boxes lead to WRITE. Experience both at Irvine and at Harvard indicates that competent secretaries quickly pick this up. During the first few sessions, someone experienced in using the terminal (perhaps a student) should be present to give advice when unusual situations arise.

A complicated dialog will have areas which cannot be transcribed successfully by the secretary. The secretary is told that if she does not understand something, she should put a row of asterisks. This serves as a marker to indicate that some editing work is needed at this point.

chart into the statements which
ese statements into the computer.
to type the statements directly
-editing program (EDIT) available
of typing is necessary even for
tice has been to use professional
ranscribing programs from flow

shown some flow charts and given
They are then introduced to the
ventions of the editing system.
pist is able to type directly from
when she sees the square boxes,
statements, and that the
nce both at Irvine and at Harvard
quickly pick this up. During
ienced in using the terminal
to give advice when unusual

which cannot be transcribed
ecretary is told that if she does
put a row of asterisks. This
ome editing work is needed at

The next step is the conversion of the secretary's material, as stored on disk, into a working program. A student assistant may be the best person to perform this task. He need not have a detailed knowledge of programming but he must understand flow charting and the use of the commands better than the typist. If problems arise which the student assistant cannot handle, he should have access to an experienced programmer. He uses the on-line editing facility to correct errors or omissions of the typists and leaves a complete program stored on disk. Although it may still contain errors, it is now a program in the form acceptable to the Metasymbol assembler.

Assembly

After the conversational program has reached this state, it must be assembled into a working (binary) program, debugged, and tested before being released for student use.

Metasymbol programs must be assembled as a batch job, either from control cards or from a terminal in the BTM system. In either case, the disk file continuing the course material will be the source input file to the assembler. The dialog facilities are kept as a "system" in the Sigma-7 Metasymbol sense; it is stored on disk under the name "DIALOG". It is this system to which the initial statement of the program refers.

Here is a procedure for assembling the program, using BTM and assuming binary output. The file with the conversation program (the source input file) is called COURSE, (you might, of course, give it a different name) and the file for keeping the binary output of the assembler is

COURSEBO (again, not a required name). The following 'cards' would perform the assembly:

```
!JOB        (accounting information--inquire locally
            for details)
!LIMIT      (TIME,5)
!ASSIGN     M:SI,(FILE,COURSE)
!ASSIGN     M:BO,(FILE,COURSEBO)
!METASYM    LS,SI,BO,AC(9999)
```

System dialog is assumed to be in account 9999. The METASYMBOL option BO and SI specify binary output and source input. LS means "list source." (Other options are available. The reader is referred to the Metasymbol manual, referenced in the Bibliography, for details.)

As with all programs of any complexity, several runs will be necessary before an error-free assembly is achieved. The assembly program will identify errors; the cause of the trouble is usually obvious once pointed out; occasionally, however, the advice of an experienced programmer may be necessary.

Running the program

The result of the successful assembly is a binary file, a program almost ready to run. This binary file can be loaded from a terminal. The authors will want to try the programs, looking for bugs, after it has been assembled successfully; copies of the flow chart and the program are valuable aids during this testing. The main branches will all have to be checked; however, testing of this kind will not discover all the bugs: only student use will do that! DELTA (an on-line following 'cards' would

tion--inquire locally

)

9999. The METASYMBOL option
input. LS means "list
e reader is referred to the
graphy, for details.)

eral runs will be necessary
The assembly program will
is usually obvious once
ice of an experienced pro-

binary file, a program
be loaded from a terminal.
looking for bugs, after it
the flow chart and the
ing. The main branches will
this kind will not discover
t! DELTA (an on-line

debugging aid on the Sigma-7) is also very useful to the experienced programmer in program checking, as it allows the user to go to any labelled statement: he need not start from the beginning or a restart entry.

The final version of the program, which will be used by the student, should be generated as a "load module." The student will then find it easy to call for it:

```
!RUN
LOAD MODULE FID: PROP
;G
```

(PROP is the name of the program; underscored characters are typed by the computer.)

When the student wants to leave the program he can follow the usual procedure, pressing the escape key twice or he can type, at any input, the word STOP. If he types STOP and if restarts are included in the program, he is reminded to use the same identification the next time he tries this dialog.

EXAMPLE OF A DIALOG

Perhaps the best explanation of the dialog system is an illustration. The fragment of a dialog on the following page is a typical example of the material which has been written using this facility. The reader may wish to examine it rather closely, for it illustrates several aspects of the system.

The first thing he may notice is that most of the commands are "WRITE"; few of the more exotic commands described in Chapter Three have been used. This is to be expected; the bulk of these programs is made up of conversations with the student and there is remarkably little other coding.

The example is indeed a fragment, chosen out of the middle of a much longer lesson. Hence it has no beginning (SYSTEM DIALOG) or end (END DIALOG). The counters (ELEC,GRAV) are defined elsewhere in the program. It should be possible for the careful reader to follow the dialog and see how it is constructed. Some additional clues are provided on the page following the dialog itself. If one follows all the paths through the segment, he will see that they all come out again at L36, which is the beginning of the next part of the program.

m is an illustration.
s a typical example
s facility. The reader
llustrates several

he commands are "WRITE";
apter Three have been
e programs is made up
remarkably little other

the middle of a much
EM DIALOG) or end
ined elsewhere in the
reader to follow the
itional clues are
f. If one follows
that they all come
next part of the


```
CONT1     WRITE     'OUR PROBLEM IS TO FIND SOME'
          WRITE        'MEASURE OF THE RELATIVE STRENGTH OF'
          WRITE        'ELECTRICAL AND GRAVITATIONAL FORCE.'
          WRITE     'DO YOU HAVE ANY SUGGESTION AS TO AN IMPORTANT'
          WRITE        'PHYSICAL SYSTEM WHERE BOTH ELECTRICAL'
          WRITE        'AND GRAVITATIONAL FORCE MUST BE'
          WRITE        'PRESENT '
          TO L24
CONT    WRITE     'YOU''RE DOING FINE'
          TO OUT
L24       RESET (ELEC,GRAV)
M1        INPUT
 IFFILTH DIRT2
M2        IF           'YES',L25
          IF           'NO',L26
          IF           'ELEC',L33
          IF           'PROT',L30
          IF        ('SUN','STAR','GALAX','THERMON'),LL25
          IF           ('HYDRO','ATOM'),L301
          BUMP GRAV
          SAVE 'SYSTEM'
          TO GRAV2,(GRAV,2)
          WRITE        'I''M HAVING TROUBLE UNDERSTANDING'
          WRITE       'YOU. WE WANT A SYSTEM THAT IS OF FUND.'
          WRITE        'AMENTAL IMPORTANCE,BUT IS STILL '
          WRITE        'SIMPLE AND EASILY ANALYZABLE.BEFORE'
          WRITE        'ANSWERING AGAIN, THINK OF FUNDAMENTAL'
          WRITE        'PARTICLES.'
          TO M1
ELEC1     BUMP ELEC
          WRITE        'THINK ABOUT ELEMENTARY PARTICLES.'
           WRITE       'TRY AGAIN'
          TO M1
L25       WRITE        'THEN TELL ME ',M1
L26       SWITCH ELEC,(ELEC1,ELEC2)
L28       WRITE        'TW'
          TO           L33
L29       WRITE        '2'
          TO L33
L30       WRITE        'PROTONS ARE A POSSIBILITY,BUT I AM'
          WRITE        'ONLY PROGRAMMED TO CONSIDER A PAIR'
          WRITE        'OF ELECTRONS.'
          TO  L36
L301     WRITE 'ATOMS ARE A POSSIBILITY BUT'
         WRITE 'I AM PROGRAMMED FOR A PAIR OF ELECTRONS.',L36
L33       WRITE        'GOOD...WE WILL WORK WITH A PAIR'
          WRITE     'OF ELECTRONS',L36
LL25      WRITE     'A SUN IS A COMPLEX THERMONUCLEAR SYSTEM'
          WRITE     'INVOLVING ALL THE FORCES. BUT WE NEED A'
          WRITE     'SIMPLER PHYSICAL SYSTEM. MAKE ANOTHER STAB'
          WRITE     'AT IT.',M1
ELEC2     WRITE        'TRY,INSTEAD OF ANSWERING NO    '
          WRITE     '... WE WANT TWO ELECTRONS.'
         SKIP 1
          TO L36
```

Line 1 and line 9: CONT1 and CONT are alternate entries to this portion of the program, depending on factors which have gone before.

Line 9: This statement means that the program is to go to CONT1 if the counter ELEC is greater than or equal to 1. IF ELEC is zero, "You're doing fine" is typed first.

Line 10. Note the use of double quotes to indicate single quotes inside a literal.

Line 12: Before entering the major set of decisions in this segment, the counters which will be used are reset to zero.

Line 13: The student is asked to respond and the response is examined for various key words.

Line 20: If none of the anticipated responses is received, the counter GRAV is "bumped" or increased and another message is written (lines 21-28.) Note that the author has chosen to SAVE all responses which did not include any of the anticipated key words, to analyze later.

Line 22: If the student has given unanticipated responses twice (when GRAV is greater than or equal to 2), no further attempt is made to get the answer out of him; he is told what the correct response is (at GRAV2).

re alternate entries to this
pending on factors which have

the program is to go to CONT1 if
r than or equal to 1. IF ELEC is
is typed first.

tes to indicate single quotes

set of decisions in this segment,
used are reset to zero.

spond and the response is
ords.

responses is received, the
r increased and another message
Note that the author has chosen
h did not include any of the
analyze later.

nanticipated responses twice
or equal to 2), no further
answer out of him; he is told
is (at GRAV2).

BIBLIOGRAPHY

This description is far from complete and offers little explanation of the requirements of the Sigma-7 software on which our system depends. The appropriate manuals for further information are as follows:

Sigma Symbol and Metasymbol Manual (900952)
Batch Timesharing Monitor Reference Manual (901577)
Batch Timesharing Monitor Users Guide (901679)

These documents are published by

Xerox Data Systems
701 South Aviation Boulevard
El Segundo, California 90245

Additional manuals describing this system are available. They contain more detailed technical information for people actually using the system. A System Users Manual assists those writing dialogs and a System Maintenance Manual advises those who may wish to modify the system and add new commands, or who wish to use the file facilities.

B23

ACKNOWLEDGEMENTS

The system was developed by Estelle Warner, Alfred Bork, David Robson, Steven Wolff, Randy Engel, Tom McGrew and Harold Deering.

Louise Healey read an earlier version of this material and offered suggestions for improvement. Steve Ashenbrenner from XDS, El Segundo, contributed important early advice. William Hoyland of XDS has supplied useful system assistance.

The project is supported by the National Science Foundation.

The facility described here is currently in use at the University of California, Irvine. We would be pleased to have others who have access to the Sigma-7 and Metasymbol make use of this system. For more information or copies of the programs, please contact

Alfred M. Bork
Physics Computer Development Project
University of California, Irvine
Irvine, California 92664