INTRODUCTION

Job Hunter is an integrated group of subroutines designed to facilitate coding programs that gather information from the typist by asking questions and processing answers. Job Hunter has the following design objectives:

1. To provide some uniformity in the Teletype page format for question-answer programs.
2. To give the typist a uniform control language with which he may alter the form of questions asked, request printing of previous answers, make corrections, etc.
3. To provide for arbitrary answer revision without restricting the structure of the answer-dependent question-answer network, and while requiring a minimum of extra thought and coding on the part of the user-programmer.

THE PHILOSOPHY OF JOB HUNTER

Job Hunter achieves its objectives as follows. All questions are numbered in a multilevel outline form under partial control of the user program. All questions are asked and all typeins occur under Job Hunter control, thus permitting Job Hunter to respond directly to commands, but to pass answers on to the user program.

A table of questions asked and answers received is built up in Job Hunter's temporary storage area (with drum buffering). Whenever the typist gives a command to change or examine the answer to a previously answered question, Job Hunter performs a "restart;" that is, it starts the user program over again at a user-program-specified beginning point. When it restarts, Job Hunter does not reinitialize its tables of questions and answers. This permits Job Hunter to answer each question itself

as the user program "asks" it. Essentially, the user program is completely reexecuted, but the Teletype input/output is simulated (by Job Hunter) except for (a) any questions specifically requested by the user to be printed or changed and (b) any dependent questions not previously asked. The latter occurs if the user changes the answer to a referenced question so as to effect a new path through the network.

Job Hunter user programs are divided into sections relating to successive questions. When the beginning of a new section (marked by an IOT) is reached, Job Hunter selects a mode in which to ask the question. Choice of mode is a function of (a) whether or not the question has been asked before, (b) the mode the previous question was asked in, and (c) indicators in Job Hunter's table set by typist commands or IOT's.

These modes and their effects are

Normal--Entered when a new question is encountered; all typeins and typeouts actually occur.

Simulate--Entered when a previously asked question is encountered and there is no reason to enter another mode. No Teletype activity occurs; the previous answer is returned to the user program. (A question is considered previously asked only if both the number and the text match an entry in the question number table.)

List--The question, the previous answer, and any comments are typed out when the appropriate IOT's are encountered, thereby producing a clean list.

Type Out for Correction--The question, the answer, and any comments are typed out; after typing the answer, Job Hunter awaits typein. The typist may enter a new answer or a special command to leave the previous one.

Null--Null mode is like simulate mode except that null answers are returned to the program at typein IOT's. Once initiated by typist command or IOT, null mode remains in effect until a previously answered question is encountered or until any type of restart occurs. This mode allows skipping a group of questions at typist or program request.

The process of asking a question and getting an answer consists of the following four steps; the first three are done by Job Hunter routines, called with IOT's.

1. Generate a question number
   This step separates sections of the program relating to different questions. The following occurs: (a) the question number is incremented, (b) an entry is made in the question-number table if no entry exists, (c) a mode is selected, and (d) the question text is specified by the user program.
2. Type out a question
   In all modes except simulate and null, this step consists of typing a Carriage Return, Line Feed, indentation, question number, and question text as specified in Step 1. The question text is available in a short and a long form; both the user program and the typist can change the printout format.
3. Typein
   In normal mode, Job Hunter awaits typed input, checks the input for typist commands, and stores the text for future reference. In all modes a pointer to the answer, whether old or new, is returned to the user program.
4. Verification and analysis of the answer by the user program.

JOB HUNTER IOT'S

Six basic IOT's have been provided to permit the user to call Job Hunter routines; four of these have variations. Where not otherwise specified, the operation of IOT's is described for normal mode, but the IOT's appear the same in other modes to the user program. Some of these IOT's take arguments; of these arguments those specified as "indirectable" are appropriately traced if the indirect bit is set. All Job Hunter IOT's are transparent to the AC, IO, and flags.

INITIALIZE JOB HUNTER (INIT)

INIT initializes Job Hunter, marks the point at which restarts are to begin, and sets up lower core registers TISMAX and TTTSU. The contents of TISMAX must not be altered by the user. An INIT must be executed before any other Job Hunter IOT, but before the INIT is executed, the six-register block of pointers at JBUFP must be set up (see section on Job Hunter storage).

THE QUESTION IOT

This IOT has numerous variations and twelve defined symbols. Its functions can be divided into (1) controlling the level of question numbering, (2) generating a question number, and (3) typing out a question. For the latter two, there are the symbols:

GQN = the IOT + 1Ø	Generate a question number
TYOQ = the IOT + 4	Type out a question
GQNT = the IOT + 14	Do both

Question Numbering

Job Hunter question numbers are printed in the form 1, 2, 3, 1A, 1B, 1C, 1A1, 1A2, 1A3, etc. There can be up to six "levels" or groups of letters or digits. The numeric levels can have

values from 1 to 63; the alphabetic levels run from A through Z, then AA, BB through ZZ, then AAA through KKK.

Level Control: +2Ø and +4Ø bits of question IOT

When the GQN function is executed, Job Hunter normally obtains question numbers by incrementing the previous number on the current level (e.g., 1B follows 1A). However, combinations of the +4Ø and +2Ø bits on the question IOT will cause one of three level-changing functions to occur before the GQN function is executed. These functions are (1) go down one level (e.g., 1A1 follows 1A), (2) go up one level (2 follows 1A), and (3) index and go down one level (1B1 follows 1A). The following symbols are defined for these options, using the prefixes D, U, and XD for the three level-controlling functions: DGQN, DGQNT, UGQN, UGQNT, XDGQN, XDGQNT.

Example: If the following IOT's are executed in order:

"GQNT", "GQNT", "DGQNT", "GQNT", "GQNT", "UGQNT", "GQNT"
"XDGQNT", "DGQNT"

the following question numbers will be typed:

1, 2, 2A, 2B, 2C, 3, 4, 5A, 5A1

It is also possible to set the +2Ø and +4Ø bits without the GQN bit being set; in this case the level change applies to the next question number generated. Three more symbols are defined for these cases: UPLV, DNLV, and XDNLV. These separate level-controlling IOT's should be located as close as possible before the GQN's they effect.

The behavior of any Job Hunter IOT (including the SICKTT IOT put in TTTSU by the INIT IOT) executed between a level change and the following GQN is unspecified.

The GQN Function: +1Ø bit on question IOT

The +1Ø bit on the question IOT is interpreted after any level-controlling function is completed. This bit causes Job Hunter to generate a question number, which means

1. Select a mode, make a question-number table entry, and remember the location of the GQN as the place to restart if the question must be reasked.
2. Increment the question number.
3. Accept an argument which must be an indirectable pointer to a block of three text pointers to the question texts: The short form, the long form, and a brief explanation.
4. Set up register JDANS with either a drum pointer to the existing answer to the question if it has been answered before, or with -Ø.

The GQN function is separate from the TYOQ function to permit the programmer to place code between the two and to allow independent control over question printing. Any code between the GQN and the TYOQ is executed in the mode established at the GQN, and it is repeated if the question is reasked.

Example: To type a question-header that should be printed wherever the question is reasked, listed, or typed out for correction.

```
        GQN                     /Generate question number
         Q1
        TYOC                    /Type header
         COMENT
        TYOQ                    /Type question
        .
        .
        .
Q1,     Q1S                     /Text pointers and text
        Q1L
        Q1H
Q1S,    TEXT /SHORT Q#/
Q1L,    TEXT /LONG FORM OF QUESTION #/
Q1L,    TEXT /BRIEF EXPLANATION #/
COMENT, TEXT /QUESTION HEADER/
```

If the question header were not needed, the following could be used:

```
GQNT
  Q1
```

with Q1 as before.

The TYOQ Function: +4 bit on question IOT

The +4 bit on the question IOT causes a question to be typed out with the question number and text specified when the GQN function was executed. The contents of the Register JBF (changed by typist commands and by the program) determine the form in which the question is typed.

0	"nothing mode"	question numbers only
1	"short mode"	short form
2	"long mode"	long form
3	"how mode"	long form and explanation

Two more bits on the question IOT control question typeout format. These bits are interpreted only when the TYOQ bit is set.

+2 means suppress Carriage Return, Line Feed, and indentation before typing question number.

+1 means suppress indentation and question number typeout.

Neither option suppresses the single space typed before the question text, and neither is executed if the previous question was asked in simulate mode.

THE TYPEIN IOT (TYIN, TYIV)

This IOT causes the program to accept input. Other Teletype input IOT's should not be used in Job Hunter programs because they are not sensitive to Job Hunter's modes. Three bits may be set on this IOT; the two more important ones determine whether the input is to come from the Teletype or from user core and whether verification is to occur or not.

Two programming considerations apply to all variations of the typein IOT. First, it must always be possible to reexecute any

code between the GQN and the typein IOT('s) because there are numerous conditions which will cause Job Hunter to reask the current question by restarting the program at the GQN. Second, if more than one typein IOT is executed per GQN, each is executed independently and appropriately for the mode established at the GQN, but only the last answer is remembered. Thus if the program encounters n Teletype typeins in list or type-out-for-correction mode, the latest answer will be listed or typed out for correction n times.

TYIN = typein IOT +Ø

This IOT causes the program to accept Teletype input as follows:

1. Program waits for typein, terminated by End-Of-Message or Rubout.
2. Input is edited. "\" deletes previous character (recursively), "Rubout" types "#" and reasks the question.
3. If the input were a typist command (see below), the command is executed. After most commands the question is reasked.
4. If the input is not a command, control is returned to the user program with the answer remembered in Job Hunter's text buffer and pointers set up as follows:

    FSA and JANS: core pointers to answer
    JDANS: drum pointer to answer

Note that the answer will not necessarily stay in core after other Job Hunter IOT's are executed; however, if the drum pointer is saved, the answer may be retrieved with the GTEXT IOT.

The drum pointer left in JDANS by TYIN will be equal to that left in JDANS by the preceding GQN if and only if no answer was typed in (i.e., Job Hunter simulated the input).

TYIV = Typein IOT +2

TYIV accepts and syntax verifies typein. It takes a single argument, an indirectable pointer to the syntax definition to be applied to the answer (see writeup on Syntax Verifier). The execution of TYIV is as follows:

1. Typein is accepted and edited; control commands are executed, and FSA, JANS, and JDANS are set up as with TYIN.
2. If the input is not a command, Syntax Verifier is called from Job Hunter.
3. If Syntax Verifier gives return 1 (illegal format), "FIX" is typed, and the question is reasked.
4. If Syntax Verifier gives return 2 (OK), control returns to the user program.
5. The answer is remembered properly only if the user's syntax definition permits Syntax Verifier to give return 2 for Job Hunter. However, if the answer is not to be remembered, either of the following may be executed without completing verification (that is, by JMPing out of the syntax definition):
    a) any variation of the kill-restart IOT which kills the current question.
    b) a typein from core.

    In these cases, verification may not be completed except after executing a typein from core without verification (TYIN+1 or TYIN+5).

TYIN+1, TYIV+1

Adding 1 to typein IOT causes a "typein from core" which differs from TYIN or TYIV as follows:

1. No Teletype activity occurs.
2. Answer is taken from user core, according to a text pointer in the AC.
3. Answer is not edited.

Control commands typed in from core are properly interpreted; answers typed in from core are remembered and simulated as answers typed in from the Teletype. In Type-Out-for-Correction mode, a typein from core is treated as in normal mode.

TYIN+5, TYIV+5

These force a typein from core to occur even when Job Hunter is simulating I-O. This option should be used with great discretion because of the danger of filling Job Hunter's text buffer with garbage. Its main application is for internally changing null answers, where a regular typein from core is sometimes inadequate because it does not occur in null mode.

GET TEXT (GTEXT)

The IOT GTEXT is called with a drum pointer in the AC to the text of the answer desired in core. This pointer must be of the form that Job Hunter leaves in register JDANS at the GQN and at the typein IOT. The IOT brings the text into core and leaves a pointer to it in FSA.

TYPE OUT COMMENT (TYOC)

This IOT must be used to type all typeouts except questions because it suppresses the typeout when Job Hunter is simulating or nulling. TYOC must be followed by an indirectable text pointer to the text to be typed. The option TYOC+1 must be followed by an indirectable pointer to a two-word block of text pointers. The first pointer is used in short and nothing modes; the second, in long and how modes.

THE KILL-RESTART IOT (RESTAR, KILL, RECONS, REASK, FAIL)

This IOT provides for modification to the question-number table and for restarts. Modifications to the question-number table include "reconsidering" questions and "killing" questions.

Reconsidering a question marks its entry in the table so that the next time it is encountered, it will be typed out for correction. Killing makes a question look as though it had never been asked before, so that it will be asked again when encountered.

Killing and Reconsidering: +1Ø, +2Ø, and +4Ø bits

The following question-number table modification options are available:

RESTAR = kill-restart IOT +Ø - No modification to question-number table. If no restart bits set, an NOP.

RESTAR+1Ø - Same as RESTAR.

KILL = kill-restart IOT +2Ø - Kills the question whose number is given in the two words following the IOT.

KILL+1Ø - Kills the question whose number is in the AC and IO.

RECONS = kill-restart IOT +4Ø - Reconsiders the question whose number is in the succeeding two words of the calling sequence.

RECONS+1Ø - Reconsiders question whose number is in the AC and IO.

REASK = kill-restart IOT +6Ø - Kills the current question, except in simulate mode when the question is reconsidered.

FAIL = kill-restart IOT +7Ø - Same as REASK except that "FIX" is typed.

The internal form for question numbers (as required by KILL and RECONS) consists of 6 bytes, one for each level. Thus, for example, the internal form of 1B is Ø1Ø2ØØ, Ø; that of 3B2CC is Ø3Ø2Ø2, 35ØØØØ.

Example: To kill question 4C, execute

```
KILL
Ø4Ø3ØØ
Ø
```

1 November 1966

Restarts: +4, +2, +1 bits

These bits on the kill-restart IOT determine what type of restart is to take place after the changes to the question-number table are completed. These bits are interpreted even if a nonexistent question number was specified with a KILL or RECONS IOT.

+Ø Don't restart - Go on.

+3 Restart at last GQN in nulling mode - If the current question is killed (as by REASK+3), this option behaves as though "←N" was typed in (see typist commands, below).

+4 Restart at INIT - Unless the program takes a new path through its question network, the program is simulated except for killed or reconsidered questions.

+5 Restart from INIT in list mode - Types out a clean listing as though "←L" typed in.

+6 Restart from calling sequence - This option takes three words of argument: an indirectable pointer to a GQN at which to restart and the question number which the GQN is to generate. These words follow any arguments required by the first part of the IOT.

+7 Restart at last GQN - Only the section of the program for the current question is reexecuted. E.g., FAIL+7 types "FIX" and reasks the current question in the same way a syntax failure at a TYIV does.

N.B., it is possible to kill and/or reconsider several questions by executing several KILL and/or RECONS IOT's, with the appropriate restart bits set only on the last one.

Example: To kill question 4 and 4A and restart from question 3 whose GQN is at LQ3:

```
KILL
4ØØØØØ
Ø
KILL+6
4Ø1ØØØ
Ø
LQ3
3ØØØØØ
Ø
```

JOB HUNTER STORAGE

A macro called JOBHUNTER is provided on the p-tape. This macro assigns the user 384Ø. characters of text storage and 128. question-number entries, using 33Ø words of the user's core. This macro also assembles a 1 into JBF. Job Hunter users have the choice of using the macro JOBHUNTER at that place in their program where the buffers are to be assembled, or of not using the macro, but instead of setting up their buffers and the 6-register block at JBUFP in lower core as they desire (see Appendix A). Note that neither Job Hunter nor the JOBHUNTER macro sets up the register IOPMAX; the user program must set IOPMAX above the buffer area whenever any Job Hunter IOT except TYOC or INIT is executed. Job Hunter programs must not use core below the symbol ORG defined on the p-tape.

TELETYPE ERRORS

Job Hunter handles all Teletype errors except breaks by hanging the user with the TYIHNG IOT. When interrupted by null (break), Job Hunter types "INTERRUPTED" and reasks the current question. Job Hunter sees breaks only when Teletype activity occurs. Although Job Hunter Teletype errors do not trap, the INIT IOT sets up TTTSU with the IOT SICKTT which is a call to Job Hunter's Teletype error routine.

TYPIST CONTROL

Instead of answering any question in the course of a question-asking program, the typist has the option of giving a command. All commands must begin with a "Back-arrow" (←). Here is the set of legal commands:

Command	Action
←/S	Type questions in short form.
←/L	Type questions in long form.
←/H	Type questions in long form followed by explanation.
←/	Don't type question, just question numbers.
←HOW	Type explanation for this question.
←←	Halt program (Transfer to register JHTSU).
←CRASH	Crash program (execute 765432).
←C#	Copy the answer to question specified (# stands for a legal question number, like 2C). The answer is printed and used as the answer to the current question.
←R#,#,#...	Type each of the given question(s) out for correction. The typist may give a new answer or ←.
←	Leave the old answer--legal only at a question typed out for correction.
←K#,#,#...	Kill and reask each of the given questions.
←#,#,#...	At present, same as ←R#,#,#...
←L	Print a clean list of the entire program since last INIT.
←L#	List specified question (print question and answer).
←L#-#	List all questions and comments between specified questions (inclusive).
←L-#	List from beginning through specified question.
←L#-	List from specified question to current question.

Combinations may be strung together with ←L commands by using commas, e.g., "←L2,4A-5B,7-".

←N	Begin nulling. A null answer is provided for current question and subsequent questions until a failure occurs or a previously answered question is encountered. Next question typed is normally the first one which will not accept a null answer.

APPENDIX A

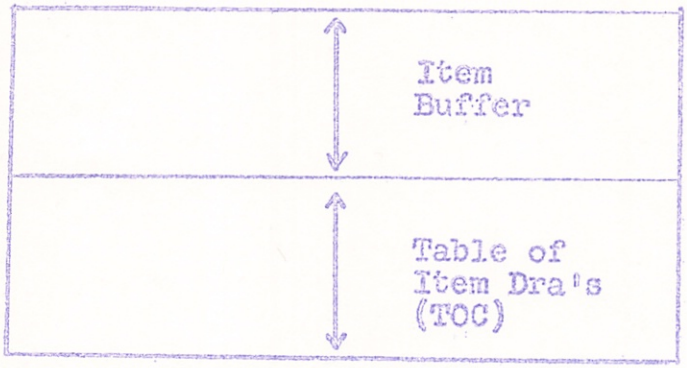
JOB HUNTER DRUM BUFFERING

Job Hunter has two variable length tables:

(1) The text buffer

(2) The question-number table

Each of these logical tables is in the format of fixed length items only one of which is in core at any given time, but which are thought of as contiguous. The drum addresses of these items are kept in core in a table of drum addresses which is built up as the logical buffer grows. When this table is filled an error comment is printed. Each logical table looks as follows in core:



With the following restrictions: The item buffer must have the item overhead (word count and rewrite number) set up initially, and the word count must be ≥49. Furthermore, the word count must be of the form 2 + 4 * N for some integer N for the question-number table. Here is a typical arrangement.

```
A,          B=A
            Ø       }   Sets up Text Buffer
A+49./B,
B+1Ø/C,
                        Program
D,          E=D     }
D+98./E     Ø       }   Sets up QN Table
E+2Ø/F,
                        Program
```

*N.B. 96. is divisible by 4.

In this case, there are B-A-2 (47.) data words in the Text Buffer Item, and room for 1Ø (8.) items, hence there's room for 8.*47.*3 characters. Also note in this case that E-D=2+4*N for N=24. That means there's room for 24 question entries (4 words) in each item in the question number table. Also there's room for 2Ø (16.) items. Hence there's room for 16.*24 question entries in the question-number table.

By leaving 1 register for the item drum address table (TOC), you prevent Job Hunter from using the drum. Note that the item and TOC table must be contiguous, but the text buffer and question number table have no set relative position.

In order to communicate the location of you buffers to Job Hunter, you assemble A,B,C,D,E, and F into 6 successive registers in lower core starting with JBUFP. The contents of these registers must be correct when the INIT is executed and may not be changed until just before the next INIT.

```
JBUFP/  A
        B
        C
        D
        E
        F
```

Job Hunter will never start a text string in one item and continue it in another. Moreover, Job Hunter's segmenting mechanism is such that it may consider the text buffer full when there is as much as 3Ø. characters of space left in it. Therefore, the buffer must be at least 12. words (including 2 for overhead) longer than the longest answer deemed legal. For example, if you arbitrarily decide that your program need not accept answers longer than 72. characters, then the minimum size for the buffer would be 12.+72./3 = 36., except that it is required to be 49. for other reasons. If we want to accept answers as long as 2ØØ. characters we will need a buffer length of 79.

APPENDIX B
SUMMARY OF DEFINED SYMBOLS

Symbol	Octal (Subject to Change)	Brief Description
DGQN	73437Ø	Down level and generate question number
DGQNT	734374	Down level, generate question number, and typeout question
DNLV	73436Ø	Down level
FAIL	73457Ø	Kill current question and type "FIX"
GQN	73431Ø	Generate question number
GQNT	734314	Generate question number and type out question
GTEXT	7346ØØ	Get text from drum
INIT	7342ØØ	Initialize Job Hunter
JANS	162	Register containing core pointer to answer
JBF	161	Brief mode flag (Ø, 1, 2, or 3)

JBUFP	15ø	6-register parameter area for drum buffering
JDANS	163	Register containing drum pointer to answer
JHTSU	2ø1	Register Job Hunter transfers to in case user types ^-
JQN	157	2-register display of current question number
KILL	734526	Kill specified question
REASK	73456ø	Kill current question (don't type "FIX")
RECONS	73454ø	Reconsider specified question
RESTAR	7345øø	NOP, but you can add restart bits
SICKTT	735ø64	Handle Teletype error
TYIN	7344øø	Type in
TYIV	7344ø2	Type in and verify
TYOC	7347øø	Type out comment
TYOQ	7343ø4	Type out question
UGQN	73433ø	Up level, generate a question number
UGQNT	734334	Up level, generate a question number, and type out question
UPLV	73432ø	Up level
XDGQN	73435ø	Index, down level, and generate question number
XDGQNT	734354	Index, down level, generate a question number, and type out question
XDNLV	73434ø	Index and down level

APPENDIX C
EXAMPLES
Example 1

This contrived example types an explanatory comment if an unsatisfactory answer is given when operating in long or how modes. The line on the typescript would appear thus if the question was #9 and if the user typed "MALE":

9 SEX MALE ONLY "M", "F", AND "U" ARE ACCEPTABLE. FIX

In short mode only the "FIX" is typed. Note that the syntax definition completes if the answer is satisfactory, but that control goes directly to the error routine at NOGOOD if the answer is not acceptable.

```
                INIT
                .
                .
                .
                (earlier questions of program)
                .
                .
                .
                GQNT
                 SQUES
                TYIV
                 SDEF
                .
                .
                .
                (rest of program)
                .
                .
                .
SDEF,           IDX SLIST               /IS IT ONE OF OK ANSWERS?
                LAC
                JMP NOGOOD              /NO
                LAC 1
SLIST,          TEXT /M###F##U###/
NOGOOD,         TYOC+1                  /TYPE COMMENT IN LONG MODE
                 BADTEX
                FAIL+7                  /TYPE "FIX", REASK QUESTION
BADTEX,         (CHARAC I#)             /SHORT MODE COMMENT IS NULL
                BADTXI
SQUES,          SQS
                SQL
                SQH
SQS,            TEXT /S#/
SQL,            TEXT /SEX#/
SQH,            TEXT /"M", "F", OR "U"#/
```

Example 2 An Answer-Dependent Question

In this piece of program, the baptism question is not asked if the answer given at the religion question was "non-Christian." Note that the baptism question is numbered on a lower level, so that the following questions have the same numbers whether or not the baptism question is asked. This is necessary to insure that the following questions will not be reasked if the religion is changed.

```
                .
                .
                .
                GQNT                    /RELIGION QUESTION
                 RELQ
                TYIV
                 RELD
                LAC OPB                 /Ø IF "NON-CHRISTIAN" ENTERED
                DAC RELGN               /REMEMBER HIS RELIGION
                SZA I                   /IS HE CHRISTIAN?
                JMP HETHAN              /NO
                DGQNT                   /YES, ASK BAPTISM QUESTION
                 BAPQ
                TYIV
                 BAPD
                .
                .
                .
                UPLV
HETHAN,         GQNT                    /SOME OTHER QUESTION
                .
                .
                .
RELD,           IDX RELS
                LAC 1
RELS,           TEXT /NON-CHRISTIAN##CATHOLIC##PROTESTANT###/
```

1 November 1966

Example 3 Internally changing a null answer
When a null answer (EOM only) is entered, "U" is typed out and typed in from core so it will appear if the question is considered or listed. Use of TYIN+5 is necessary to cause typein to occur in nulling mode.

```
                    .
                    .
                    .
                    GQNT
                     QUES                  /POINTER TO QUESTION PTR BLOCK
                    TYIV
                     DEF
                    .
                    .
                    .
DEF,                Ø                     /EOM
                    JMP NUL
                    LAC
                    .
                    .                     /REST OF DEFINITION
                    .
                    LAC 1
NUL,                TYOC
                    (FLEXO U#X)           /TYPE "U" UNLESS NULLING
                    LAW (FLEXO U#X)
                    TYIN+5                /TYPE IN FROM CORE IN ANY CASE
                    LCH I FSA             /POINT FSA AT EOM
                    STVW
```

Example 4 Avoiding Unnecessary Verification
Comparing JDANS before and after a typein is a suggested means of avoiding repetition of time-consuming verification, drum operations, etc., when the program simulates or lists or when the user reconsiders but does not change an answer.

```
                    .
                    .
                    .
                    GQNT
                     QUES
                    LAC JDANS             /-Ø or DRUM PTR TO PREVIOUS ANSWER
                    TYIN or TYIV          /IF TYIV, SYNTAX DEF TRANS TO AC
                    SAD JDANS             /IS NEW DRUM PTR SAME?
                    JMP DONT              /YES, SKIP SOME STUFF
                    .
                    .          Section of code to execute
                               only if answer changes
DONT,               .
                    .
                    .
```