THE INTERNALS OF ALGOL 205

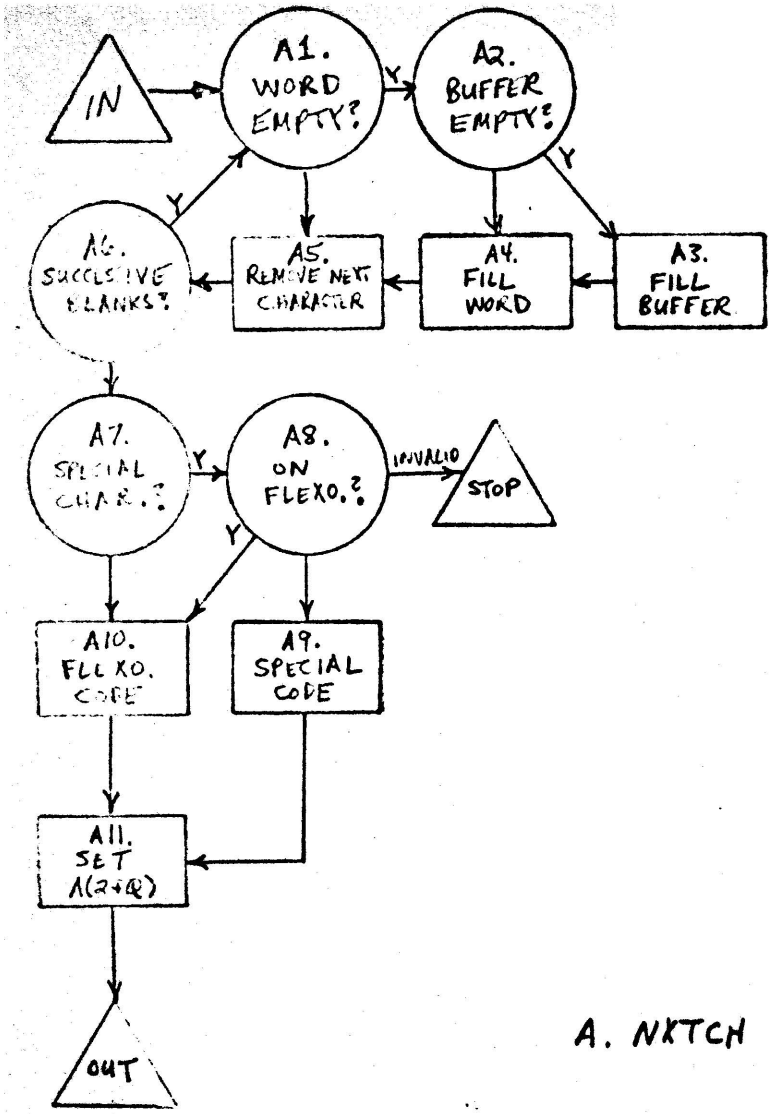
How does this compiler work? Frankly, it's a miracle if it does. The comments given here, however, serve as some sort of a key to understanding the internal mechanism of this translation process. They are not intended to be merely a restatement of what the program does, step-by-step: the symbolic listing of the program gives a precise version of that. These notes are abstracted one level from the program itself and they attempt to tell what is happening from an overall standpoint.

The reader who encounters these notes for the first time is advised to write a simple little Algol program and try to follow it through the flowcharts. Then when he gets a feel for what is happening he should take closer looks at the steps.

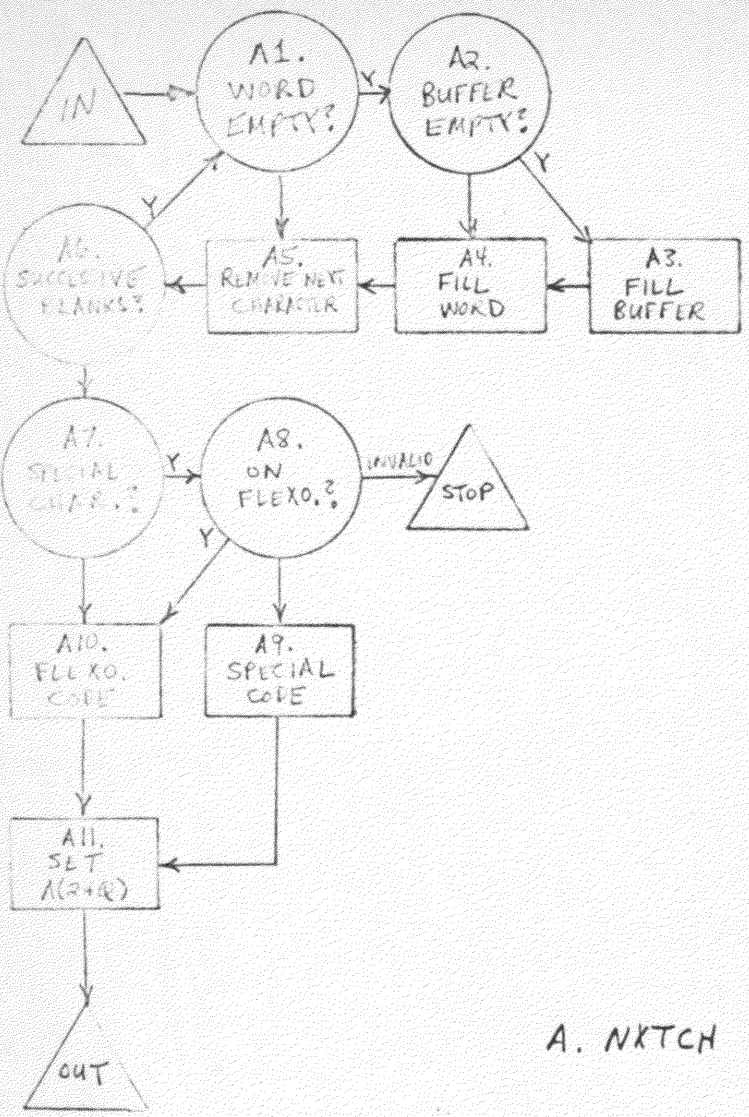
The order of presentation here is supposed to be such that the first routines illustrated use no other routines internally, the next ones use these first ones, and so on, each routine using only those which have been discussed earlier. This order was actually impossible to follow in the case of the "GET" and "MASTR" subroutines, which use each other, but elsewhere it's pretty much all right.

Some of the author's terminology may need explanation: a "counter" is a storage location which keeps count of something or other. Counters are distinguished from ordinary temporary storage in that they can be used to store only one type of information throughout the program, while temp. storage is a relatively short term storage which can be used for many things. A "stack" is another word for a table which initially contains nothing; but things get piled into it, one on top of another; and these things are generally only taken away when they are removed from the top. Several stacks are distinguished in this compiler: the operator stack, the operand stack, the subscript stack, the constant stack, the true-false stack, and the do-stack. The operation of these stacks will become clear by working through some examples. Phrases such as "put onto the top of the operator stack" and "removed from the top of the subscript stack" will be used in this description, and they should have a clear meaning by the analogy above.

The style of presentation is adapted from the practice of computer publications in the U.S.S.R.: the flowchart boxes contain only brief comments and code numbers which reference the text, so the diagrams indicate the "topology" of the situation; the explanatory text which accompanies the diagrams tells what goes on inside each box. Code numbers, a letter followed by an integer, serve to cross-reference the text, the flow-charts, and the symbolic listing of the program. The information labeled "Coding Details" is to be bypassed on first reading -- it applies directly to the symbolic machine language and are included here only for reference in partial explanation of the coding, for someone who would like to make changes.



A. NXTCH SUBROUTINE



A. NXTCH SUBROUTINE

More general information about the compiling process may be found in section I, where the program itself begins.

A. "NXTCH": Next-Character Subroutine.

This serves as the sole communication link between the compiler and the input pseudo-code. Two routines are necessary, one for Cardatron input and one for paper-tape input. It also is tied in with the ALARM routine, as it prepares the input for possible Flexowriter typeouts. Remarks: The counter WORD is initially empty, as is the "input buffer" and the "ALARM buffer." Input to this routine is the counter Q which is either 0, 1, or 2. The subroutine yields as output the next character, from left to right, of the user's program.

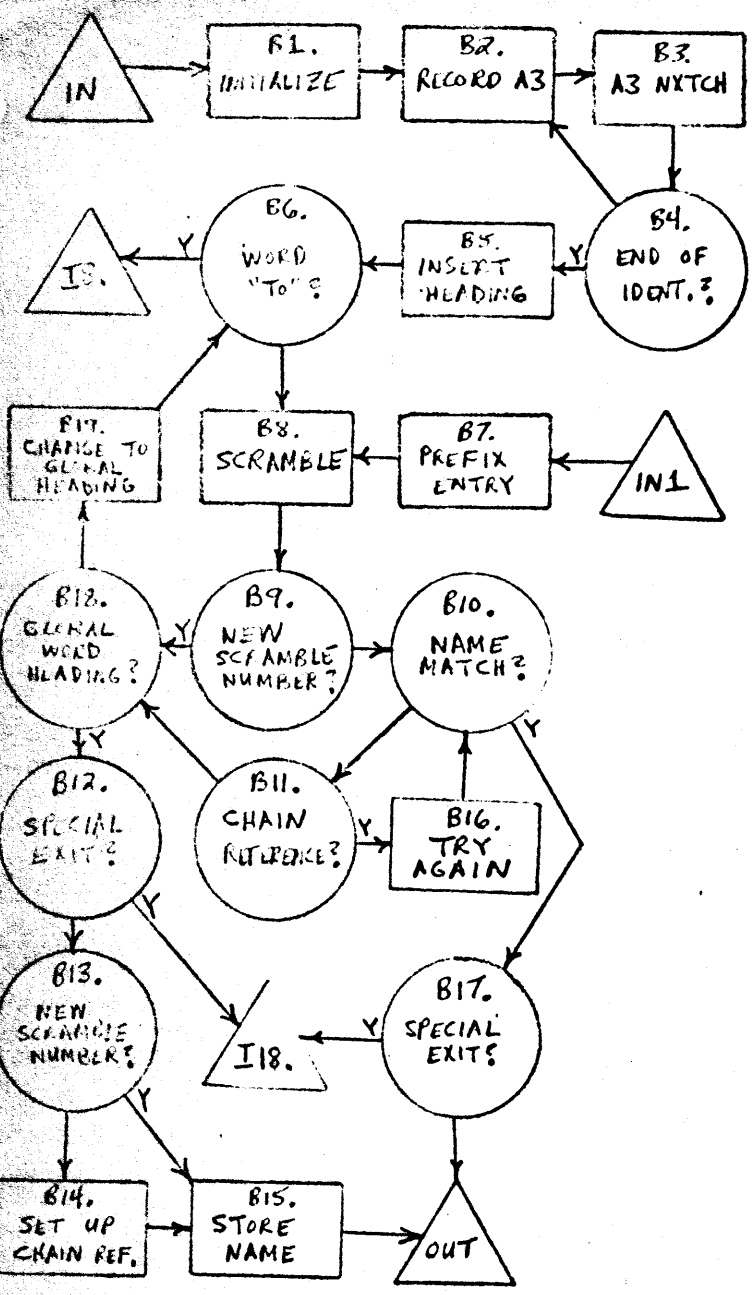
A1. If WORD is empty, go to A2; else to A5.
A2. If the input buffer is empty go to A3, else to A4.
A3. Fill input buffer with next card or next paper tape record. Stop if a paper tape record too long is sensed.
A4. Fill WORD with next five alphanumeric characters.
A5. Take next character from WORD.
A6. If both it and the last character were blank, return to A1.
A7. If it is alphabetic or numeric, go to A10.
A8. Stop if it is not a valid special character. Go to A10 if we are processing part of a Format string between asterisks. Otherwise go to A9 if the Flexowriter cannot type this character, to A10 if it can.
A9. Push space-letter-space into right side of ALARM buffer. To A11.
A10. Push it into right side of ALARM buffer.
A11. Put it into the counter A(2+Q). Exit.

Coding Details. Entry NXTCH: rA=exit, rB=desired value of Q.
Entry NXCH1: rA=exit. Q will be set to 1.
Exit: rA,LSTCH, and A(2+Q) all contain next character. rB=Q. Special characters have been translated into a code number between 1 and 10, unless inside Format strings.

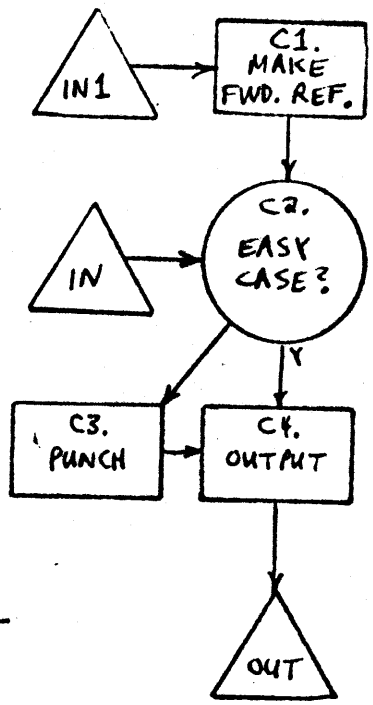
Temp Storage Used: loop 6, WORD, Q, A4, A(2+Q),Z,CURNT table, LSTCH,NEXIT.

B. "BUILD."

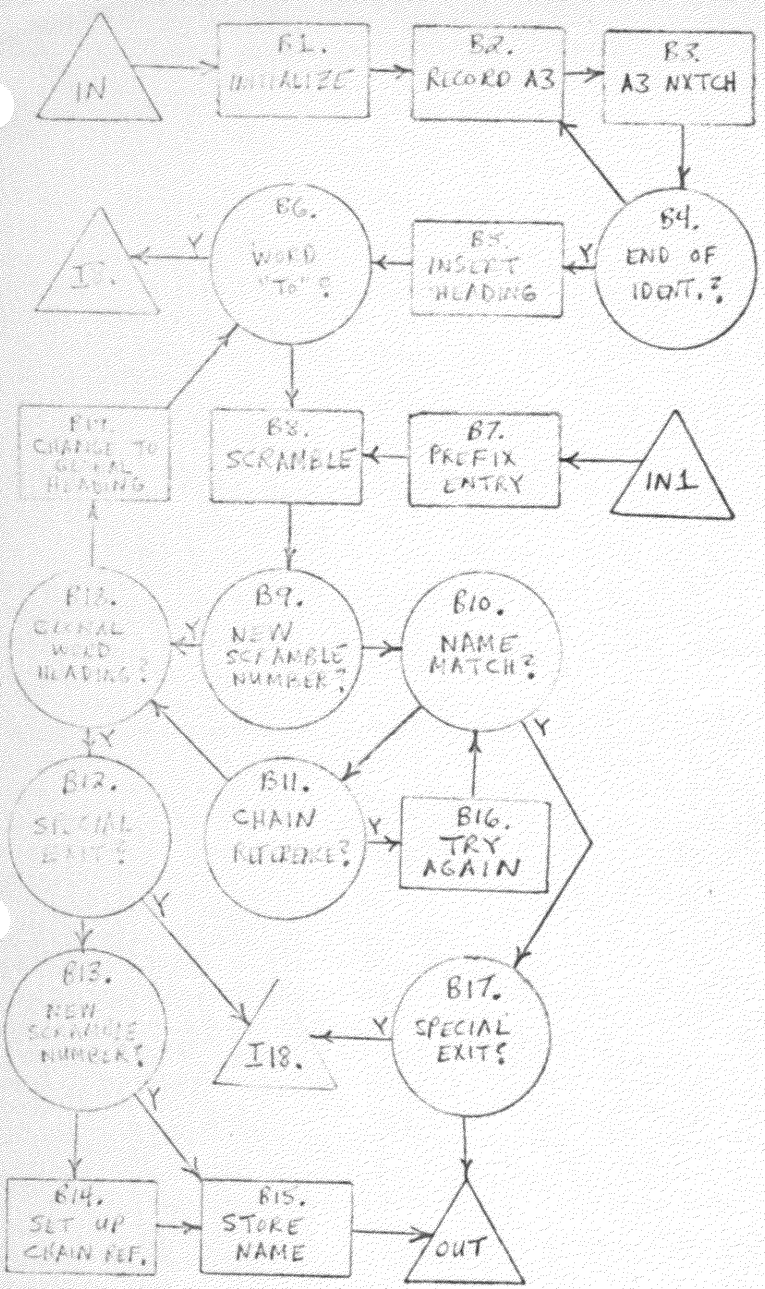
This subroutine obtains an identifier from the NXTCH routine and converts it into a number between 1 and 225. The latter number is the code number which the compiler works with. Remarks: Input to this routine is the first (alphabetic) character of the identifier in temp storage location A3. Output is the value of I.



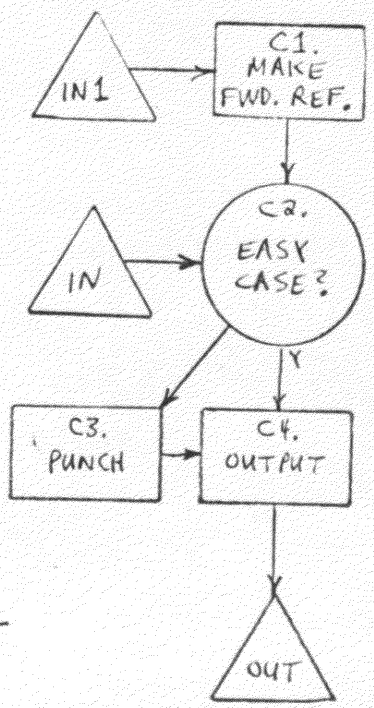
B. BUILD SUBROUTINE.



C. PUNCH SUBROUTINE



B. BUILD SUBROUTINE.



C. PUNCH SUBROUTINE

B1. Set Name Buffer to blanks and set the special flag off.
B2. Insert A3 into name buffer.
B3. NXTCH into A3(Subroutine A with Q=1).
B4. If A3 is alphabetic or numeric, go back to B2.
B5. Insert length of identifier string and procedure heading number (zero if outside procedure declarations) into name buffer.
B6. Special exit to I8 if the identifier is the word TO. Else to B8.
B7. This special entry point sets up for a lookup on only the first character in the name buffer, with its procedure heading, as a prefix character.
B8. "Scramble" name buffer into a number I between 1 and 100.
B9. If table entry for I is unused, go to B18.
B10. If identifier for table entry I matches the name buffer, go to B17.
B11. If table entry I indicates a "chain reference" to another number I, go to B16. Else to B18.
B12. Special exit to I18 if called for.
B13. IF table entry for I is unused, go to B15.
B14. Find highest unused place in table. ALARM if none are left. Put this as a chain reference into table location I and then set I to this number.
B15. Put name buffer into table associated with I. ALARM if table is packed. Exit.
B16. Set I to the chain reference. To B10.
B17. Special to I18 if called for; else normal exit.
B18. If flag is on, go to B12.
B19. Set flag on. Insert global-word heading in place of procedure heading number in name buffer. Go to B6.

Cdding Details. Entry BUILD: rA=exit. A3=first char. of ident.
Entry BLD1: EXIT0=zero if special exit desired at steps B12, B17; else EXIT0=exit. R contains ident; a prefix lookup is done.
Entry BLD2: same as BLD1, with rA=0.
Normal exit: TEMP3 contains the equivalent in the low-order 4 positions.

Temp Storage Used: TEMP1,TEMP2,TEMP3,TEMP4,MEXIT,RIGHT,LEFT,OP, GEXIT,TOGET,TODO,MFL,PVAR,SPEC,COLUM,A3,EXIT0,C1,C2,PART1 table, PART2 table, PART3 table.
Subroutine Used: NXTCH.

C. PUNCH Subroutine. This is the only link between the compiler and the punching of the output, except possibly for the initialization and FINISH where special things are punched. Two routines are necessary, one for Cardatron, one for papertape. Remarks: Input to this routine are a computer word and the location into which it is to be loaded; or, a forward reference table entry is produced.

C1. Set Location to the next available place for a forward reference table entry, set Number to the forward reference code desired.
C2. (Cardatron) If Location is one higher than previous location or if the Card Buffer area is not full, go to C4.

C3. (Cardatron) Punch contents of Card Buffer area, set Location as beginning location of new card buffer load.
C4. (Cardatron) Put Number into Card Buffer area. If Skip Switch is off, print Location and Number on 407. Exit.
C2. (PTape) If Location is one bigger than previous location, go to C4.
C3. (PTape) Punch "PTR Location" with sign of 4.
C4. (PTape) Punch Number. Exit.

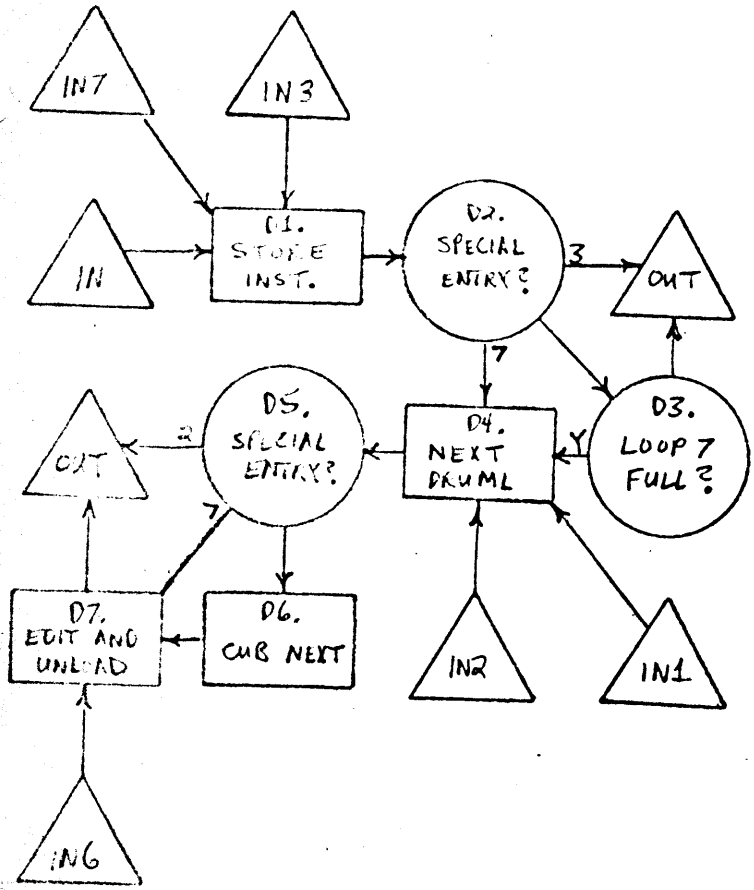
Coding Details: Entry PUNCH: rA=exit,TEMP1=Location,TEMP2=Number.
Other entrances are for forward reference entries:
Entry PNCH1: rB=exit,rR:44=forward reference spot, DRUML+LOOPL=current location.
Entry PNCH2: rB=exit, fwd. ref. entry in TEMP2.
Entry PNCH4: same as PNCH1, except rA:04=fwd.ref. spot.
Entry PNCH5: rB=exit, fwd. ref. entry in rA.
Temp Storage Used: TEMP1,TEMP2,FREFL,LLOC,loop 6,PCH table,PEXIT.

D. OUTPT Subroutine.

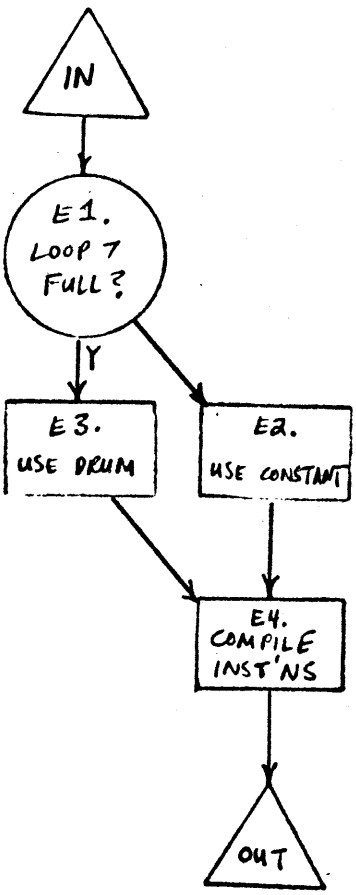
This subroutine is the normal outlet for compiled instructions; it enables the automatic use of high speed loop 7 in the object program. Remarks: DRUML is the counter which tells where the current loopfull will go start on the drum. LOOPL is the position in the buffer where the next compiled instruction will go; thus the sum, DRUML+LOOPL is the location of the next instruction to be compiled. ABC is a counter which keeps track of the number of constants which are to go into this loopfull.

D1. Put coded instruction into Loop7 Buffer.Increment LOOPL.
D2. If special entrance 3 was used, exit. If special entrance 7,to D4.
D3. If there is room for more than one more instruction in Loop7 Buffer, exit.
D4. Calculate address of next loopfull. If it comes out between 981-999,1981-1999,2981-2999, round up to next 1000.
D5. If special entrance 2, exit. If special entrance 7, to D7.
D6. Put CUB(next loopfull) into Loop7 Buffer.
D7. Edit instructions [illegible] in Loop7 Buffer from compiler code to 205 code and punch. Punch all constants for this loopfull. (Subroutine C). Set DRUML to next, and set LOOPL,ABC to zero, then exit.

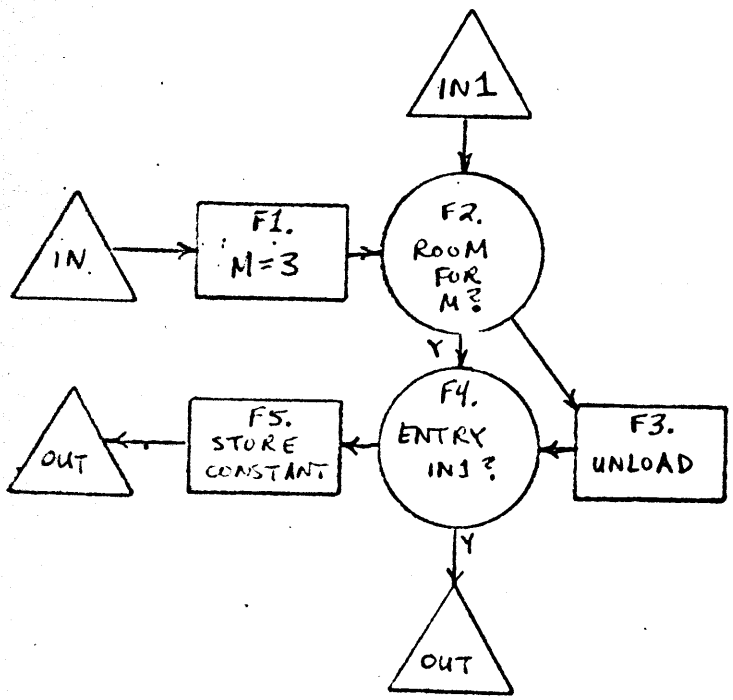
Coding Details:
Entry OUTPT: rA=exit.TEMP1=coded instruction
Entry OUTP3: same. Omits the test for full loop.
Entry OUTP5: same as OUTPT except EXIT0=exit.
Entry OUTP1: rA=exit. Acts as if loop was almost full.
Entry OUTP2: calculates address of next loopfull only.
Entry OUTP6: EXIT0=exit,TEMP3=addr. of next loopfull. Punches the loop only.
Entry OUTP7: rA=exit, TEMP1=coded instruction. Combines features of OUTP3,OUTP2,OUTP6. Upon exit, rA=0.
Temp Storage Used: TEMP1,TEMP2,TEMP3,TEMP4,DRUML,LOOPL,ABC,OUT table, CON table,LLOC,PART2 table,EXIT0.
Subroutine Used: PUNCH.



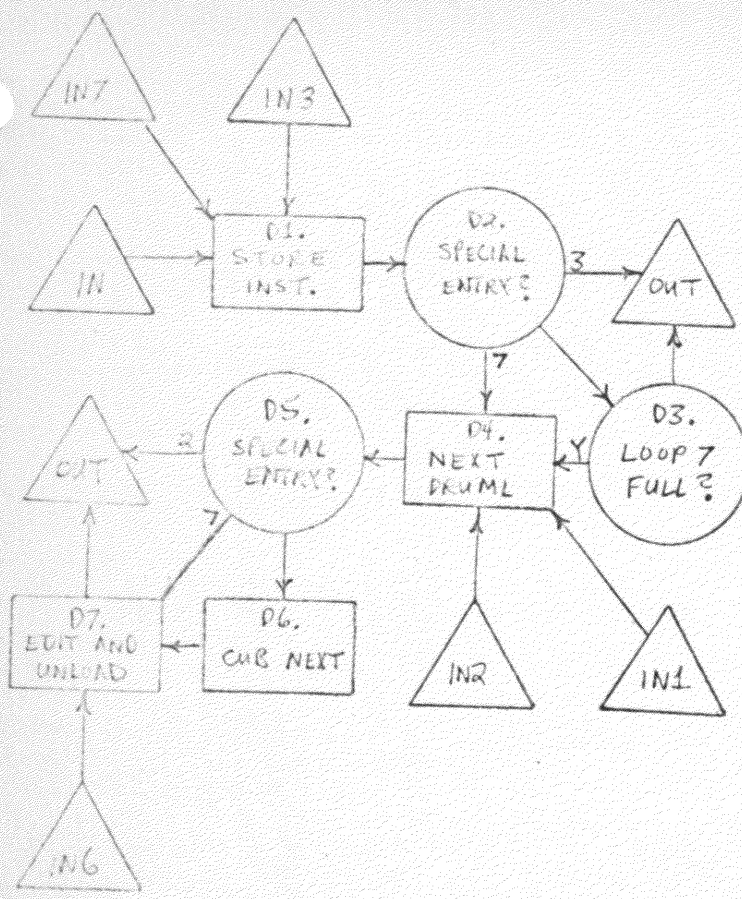
D. OUTPUT SUBROUTINE



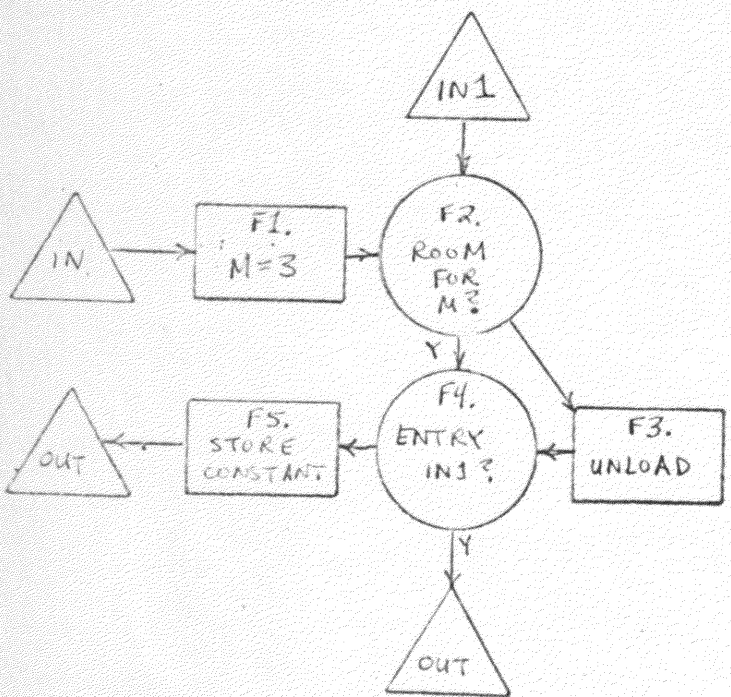
E. ACALL, BCALL SUBROUTINE



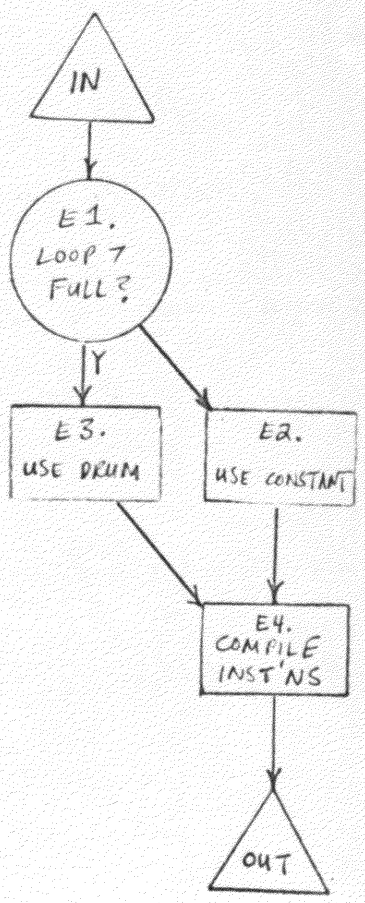
F. CONOT, FULL SUBROUTINES.



D. OUTPUT SUBROUTINE



F. CONOT, FULL SUBROUTINES.



E. ACALL, BCALL SUBROUTINE

E. ACALL, BCALL Subroutines.

These produce the object program instructions to call a subroutine with exit in the A or B register, respectively. Remarks: input is the address of the subroutine; a special flag is set by the "FOR" processor to cause the output program to jump back to the incrementation phase, otherwise control of the object program would continue in sequence, after execution of the subroutine.

E1. If there is room for only two more things in the Loop7 buffer, go to E3.
E2. Compile CAD or LDB with next constant(Subroutine D). To E4.
E3. Compile CAD or LDB with next drum location (Subroutine D, IN3).
E4. Compile CUB(subroutine) (Subroutine D, IN3). Get the address of the next loopfull1 (Subroutine D, IN2). Prepare constant CUB next, or if FOR flag is set, CUB incr. Add it to the constant buffer. Punch out the Loop7 buffer (Subroutine D entry IN6). Exit.

Coding Details: Entry: rA=subroutine address,rB=exit.If rB is zero, exit will be made to FORIN.
Temp Storage Used: TEMP1,TEMP3,TEMP4,LOOPL,ABC,DRUML,RB,CON table, PHOR1,EXIT0.
Subroutine Used: OUTPT.

F. CONOT and FULL Subroutines.

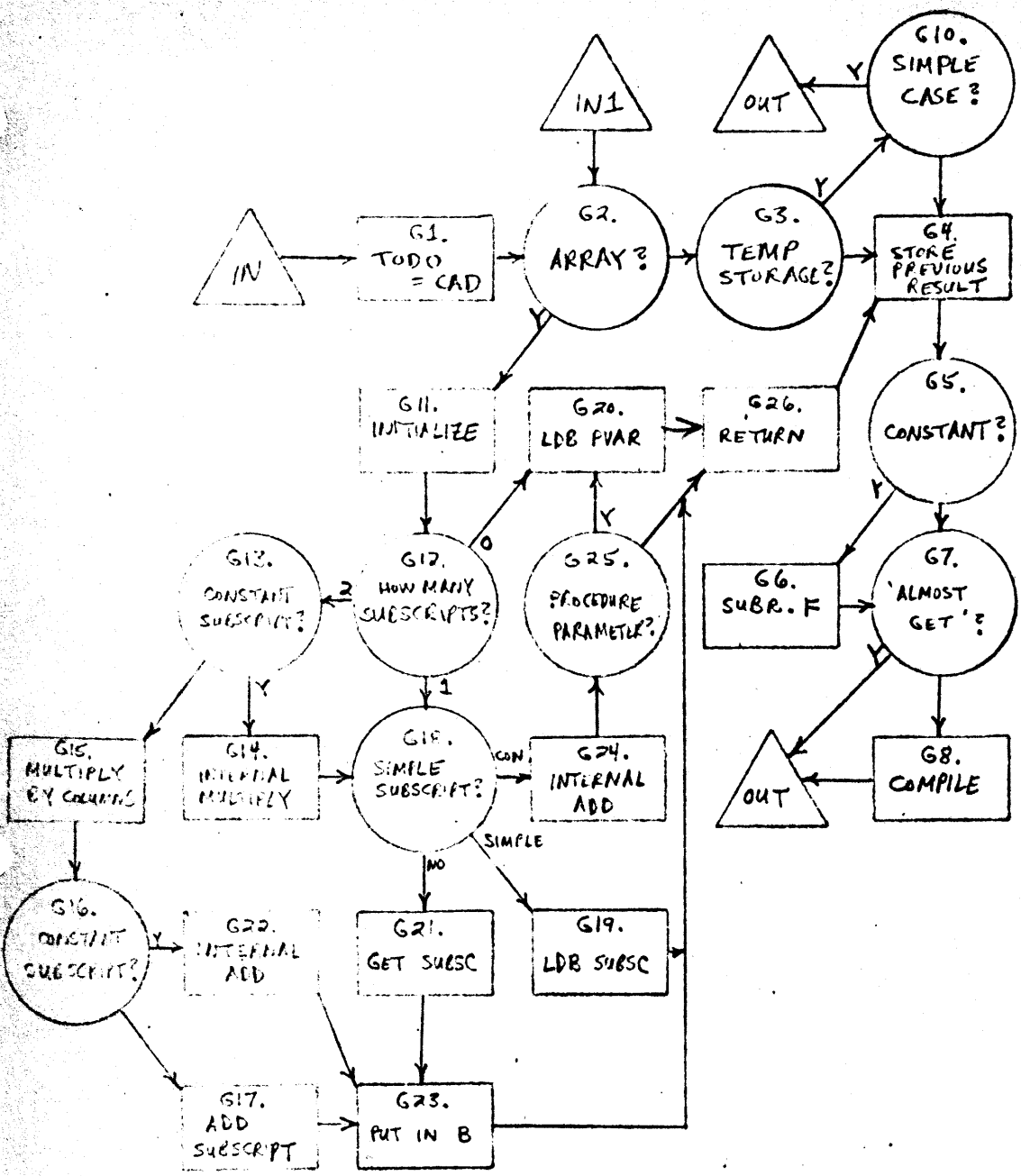
These two short routines are used fairly often. CONOT prepares a constant for compilation. FULL makes sure there is room for M more instructions in the Loop7 Buffer.

F1. (CONOT entry) Set M = 3.
F2. (FULL entry) If there is room for M more instructions in the Loop7 Buffer, go to F4.
F3. Punch contents of Loop7 Buffer (Subroutine D, IN1)
F4. If doing the FULL subroutine only, exit.
F5. Increase ABC by 1, put constant into constant buffer. Exit.

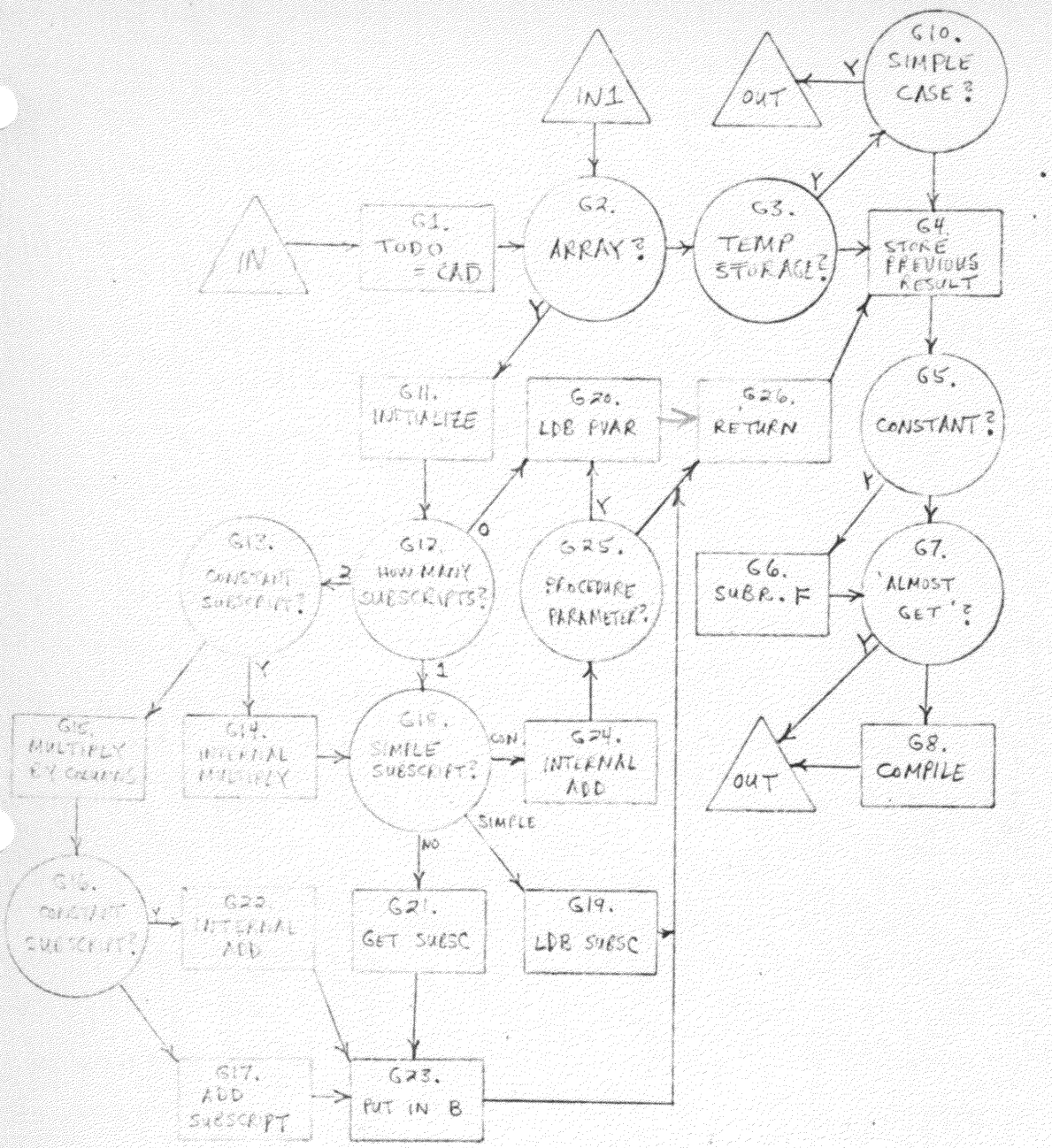
Coding Details: Entry CONOT: rA=constant, rB=exit.
Entry CNOUT: TEMP5=constant, rB=exit.
Entry FULL: rA=M-2, rB=exit.
Entry FULL1: rA=M-3, rB=exit.
Temp Storage Used, FULL: none. Subroutine Used: OUTP1.
Temp Storage Used, CONOT: EXIT1,TEMP5,ABC. Subroutine Used: FULL.

G. GET Routine.

Now we get into the meat of the compiler. This very general routine compiles the instructions to bring a coded quantity into one of the registers. It is practically the only place arithmetic instructions are compiled. The routine also does some of the bookkeeping for allocation of temporary storage. Remarks: This routine uses itself as a subroutine, and it also uses MASTR which uses GET as a subroutine, so it must store away its exits and inputs to prevent them from being overlaid. Inputs to GET are "TOGET," a code for the quantity to be compiled for, and "TODO", a code for the type of operation to compile it with. When we say merely "GET something" we mean TODO = CAD.



6. GET ROUTINE.



G. GET ROUTINE.

G1. Set TODO = CAD.
G2. If TOGET is an array, go to G11.
G3. If TOGET is a temp storage code, go to G10.
G4. If rA in the output code contains something that must be stored away, compile a store instruction (Subroutine D).
G5. If TOGET is not a constant, go to G7.
G6. If TODO = CAD and the constant is zero, compile a STC instruction (Subroutine D). Otherwise take the constant off the constant stack and execute Subroutine F (CONOT).
G7. If special "ALMOST GET" exit is called for, exit.
G8. Compile the desired TOGET-TODO instruction by using the following operation code:

TOGET		TODO							
Minus Tag	Absolute Tag	CAD	ADD	FAD	MUL	DIV	FMU	FDV	LDB
OFF	OFF	CAD	ADD	FAD	MUL	DIV	FMU	FDV	LDB
ON	OFF	CSU	SUB	FSU	MUL	DIV	FMU	FDV	LDB
OFF	ON	CAA	ADA	FAA	not used		FMA	FDA	LDB
ON	ON	CSA	SUA	FSA	not used		FMA	FDA	LDB

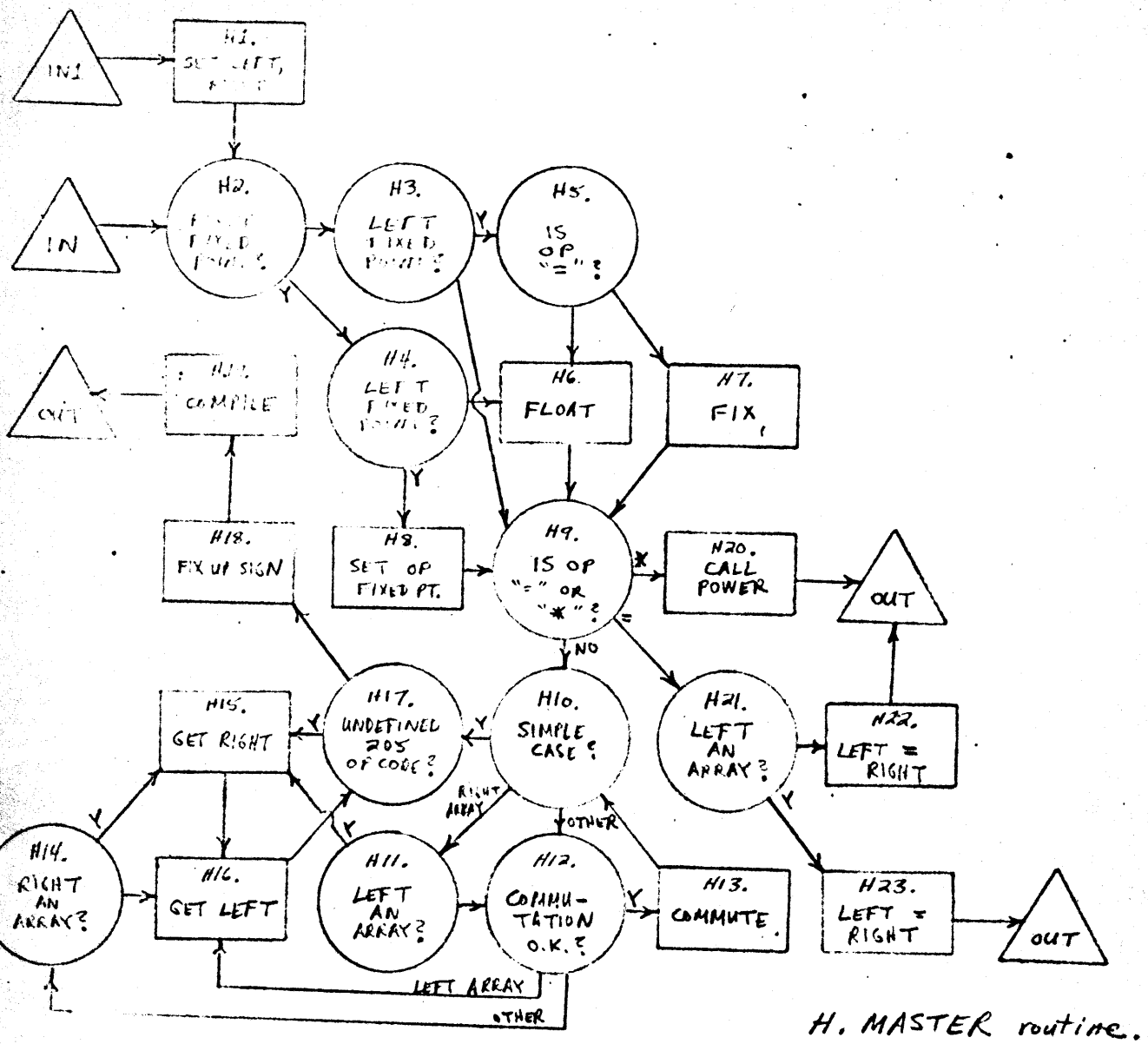
(Subroutine D).(These are the 220 mnemonics for the op-codes). Exit.
G9. undefined
G10. If TOGET is already in rA in the output, and TOGET is CAD, exit. Else to G4.
G11. Stash away inputs and temp storage.Take subscripts off top of subscript stack. If TOGET is not a procedure parameter, set SPEC to base address, set PVAR to zero. Otherwise set PVAR to input information address and set SPEC to zero.
G12. If TOGET is a vector, go to G18. If TOGET is a prodedure output variable, go to G20. Otherwise TOGET must be a matrix.
G13. If TOGET is a non-procedure-parameter matrix and its first subscript is a fixed point constant, go to G14, else to G15.
G14. Pull constant off constant stack, multiply it by hhe number of columns, and add it to SPEC. To G18.
G15. Compile to multiply the first subscript by the number of columns (Subroutine H).
G16. If the second subscript is a fixed-point constant, go to G22.
G17. Compile to add the second subscript to the previous result. (Subroutine H). Go to G23.
G18. If subscript is a fixed point constant, go to G24. If it is floating point or TOGET is a procedure parameter, go to G21.
G19. Compile to LDB with the subscript (Subroutine G). To G26.
G20. Compile to LDB with PVAR (Subroutine G). To G26.
G21. GET the subscript (Subroutine G). To G26.
G22. Take the constant off constant stack and add it to SPEC.
G23. If the result in rA is floating point, compile FAD(5810000000) or FSU(5810000000) according as rA contains the true result or its negative (Subroutines F,D). Then if TOGET is a procedure variable, also compile ADD or SUB (PVAR). Then compile STA 4001, LDB 4001 (Subroutine D over and over).To G26.
G24. Take the constant off constant stack and add it to SPEC.
G25. If TOGET is a procedure parameter, go to G20.
G26. Change TOGET to SPEC, with or without B modification. Restore temporary storage. Go to G4.

Coding Details: Entry GET: rB=exit, rA=TOGET(TODO will be set to CAD)
Entry GET1:rB=exit,TOGET=TOGET,rA=TODO.
If TODO negative, we take special "almost get" exit.
Temp Storage Used: TOGET,TODO,GEXIT,STOR,TEMP4,TEMP5,COUNT,CNO, NUMST table,SPEC,PVAR,COLUM,HOLD table,CMTX,LEFT,RIGHT,OP,RB, sometimes TEMP6.
Subroutines Used: CONOT, GET, MASTR, RSULT, STLDB.
(Note: When the temp storage is stored and restored again, the entire loop 5 is stored. Therefore all counters in this loop - Q,WORD,LSTCH,A0-A4, were chosen so that they would not change during the time the temp storage was down on the drum. Something like CNO, CMTX, STOR, LOOPL must not be kept in loop 5.)

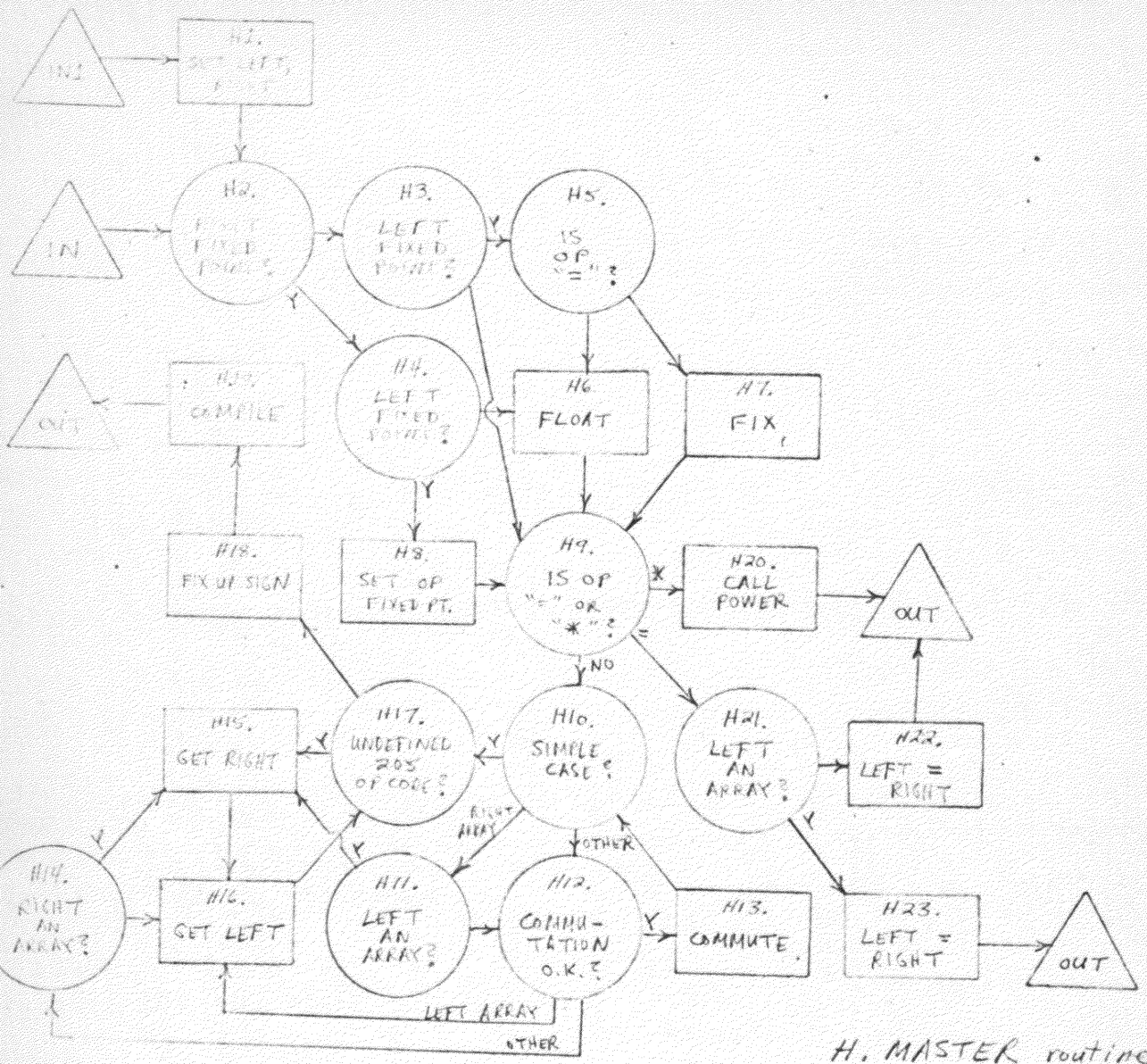
H. MASTR routine.

This general purpose routine takes charge of compiling the instructions necessary to do addition, subtraction, multiplication, division, exponentiation, and substitution. Remarks: this routine is a little tricky because it uses GET as a subroutine, and GET uses MASTR as a subroutine, and MASTR in turn uses GET yet again. The process is rigged, however, so that the nesting doesn't go any deeper than this. Inputs to MASTR are an operation OP and the operands LEFT and RIGHT. OP is thought of as being floating point unless changed to fixed point by the MASTR routine. The coding turned out by MASTR is efficient in most respects. The major inefficiency is in the treatment of arrays with very simple (e.g. constant) subscripts, when occasionally store instructions are compiled in anticipation of a difficult array subscript. The other weakness is an fixed point multiplication when LEFT has an absolute value to be taken and both LEFT and RIGHT are simple variables - this case was of such infrequent occurrence it was not accommodated.

H1. The top of the operand stack is removed and placed in RIGHT. Then the (new) top of the operand stack is removed and placed in LEFT.
H2. If RIGHT is fixed point, go to H4.
H3. If LEFT is fixed point, go to H5; else to H9.
H4. If LEFT is fixed point, go to H8; else to H6.
H5. If OP is "=", go to H7.
H6. We have one fixed point operand and one floating point operand. We want to float the fixed one; if LEFT is the fixed one and both RIGHT and LEFT are arrays, we first GET RIGHT (Subroutine G) and replace it by a temp storage code so the subscript stack will be last-in-first-out. If the operand to be floated is a constant, we merely change the constant to floating point. Otherwise we reserve room for the float subroutine in case it has not been reserved, and finally GET the fixed-point operand (Subroutine G) and BCALL the float subroutine (Subroutine E). To H9.
H7. Same as H6 except we are changing RIGHT from floating point to fixed point and there's no test for constants. To H9.
H8. Set OP as fixed point.
H9. Now the preliminaries are all taken care of and we are ready to do the real work. If OP is "=", go to H21; if OP is "*",



H. MASTER routine.



H. MASTER routine.

H10. If RIGHT is an array, go to H11. If LEFT is already in rA except possibly for its sign, go to H17. Else to H12.
H11. If LEFT is also an array, go to H15.
H12. If OP is non-commutative, go to H14. If LEFT is an array, commutation will do no good, so go to H16.
H13. Set OP as non-commutative. Interchange LEFT and RIGHT. TO H10.
H14. If RIGHT is not an array, go to H16.
H15. GET RIGHT (Subroutine G). Change RIGHT to a temp storage code.
H16. GET LEFT (Subroutine G). Change LEFT to a temp storage code.
H17. Now we have ± LEFT in register A and RIGHT is sufficiently simple to finish the operation, unless OP is fixed point multiply or divided and RIGHT is to be taken with absolute value. In this unfortunate case, go back to H15.
H18. If register A contains -LEFT, negate the sign of RIGHT. (If OP is +, the sign of the compiled result will be the sign of LEFT, otherwise it is the sign of RIGHT.) Now if OP is divide, compile either CLR (floating point) or SRT 10 (fixed point) (Subroutine D).
H19. GET RIGHT with the proper OP code (Subroutine G, IN1). If OP is fixed point multiply, compile also SLT 10. Exit.
H20. Reserve room for the power subroutine in case it hasn't been used yet. Then GET RIGHT (Subroutine G). Compile STA 4000 (Subroutine D). GET LEFT (Subroutine G), BCALL the power routine (Subroutine E). Exit.
H21. If LEFT is an array, go to H23.
H22. GET RIGHT (Subroutine G). Then, if LEFT is not the name of a procedure currently being defined compile STA LEFT (Subroutine D). Exit.
H23. If RIGHT is also an array, GET RIGHT (Subroutine G) and replace it by a temp storage code. Then "almost get" LEFT (Subroutine G, special exit at step G7). Then GET RIGHT (Subroutine G) followed be STA (LEFT) (Subroutine D). Exit.

Coding Details: Entry MASTR: rA=exit; RIGHT,LEFT,OP are set up.
Entry MAST1: rB=exit; rA=OP; RIGHT,LEFT will be set up from NSTAK.
Entry MAST2: same as MAST1 except rB will be set to exit to FIXT1
Entry MAST3: rA=RIGHT;rB=CNAME-1;OP,MEXIT are set up; LEFT will be set up from NSTAK
Entry MAST4: rA=zero;RIGHT,LEFT,OP,MEXIT are set up.

Temp Storage Used: MEXIT,MFL,RIGHT,LEFT,OP,SEG1,SEG2,HIGHL,NUMST table, T,STOR,TEMP1,COLUM,PVAR.

Subroutines Used: GET,BCALL,FIXT3.

Coding Details on "hidden" subroutines FIXT and RSULT:
Entry FIXT2: sets TMIN=min(TMIN,T) and sets T=max(COUNT+1,T). Exits to HIER.
Entry FIXT3: same except rA=exit.
Entry FIXT5: rA=exit. Sets TMIN only.
Entry RSULT: sets STOR=T=T-1, on exit rA=STOR with arithmetic of OP and sign of LEFT, rB=CNAME.
Entry FIXT1: FIXT2 followed by RSULT followed by putting (rA) on top of operand stack, exits to HIER.
Entry FIXT4: FIXT1 with rA=OP,LEFT will be set positive

I. PRESCANNER.

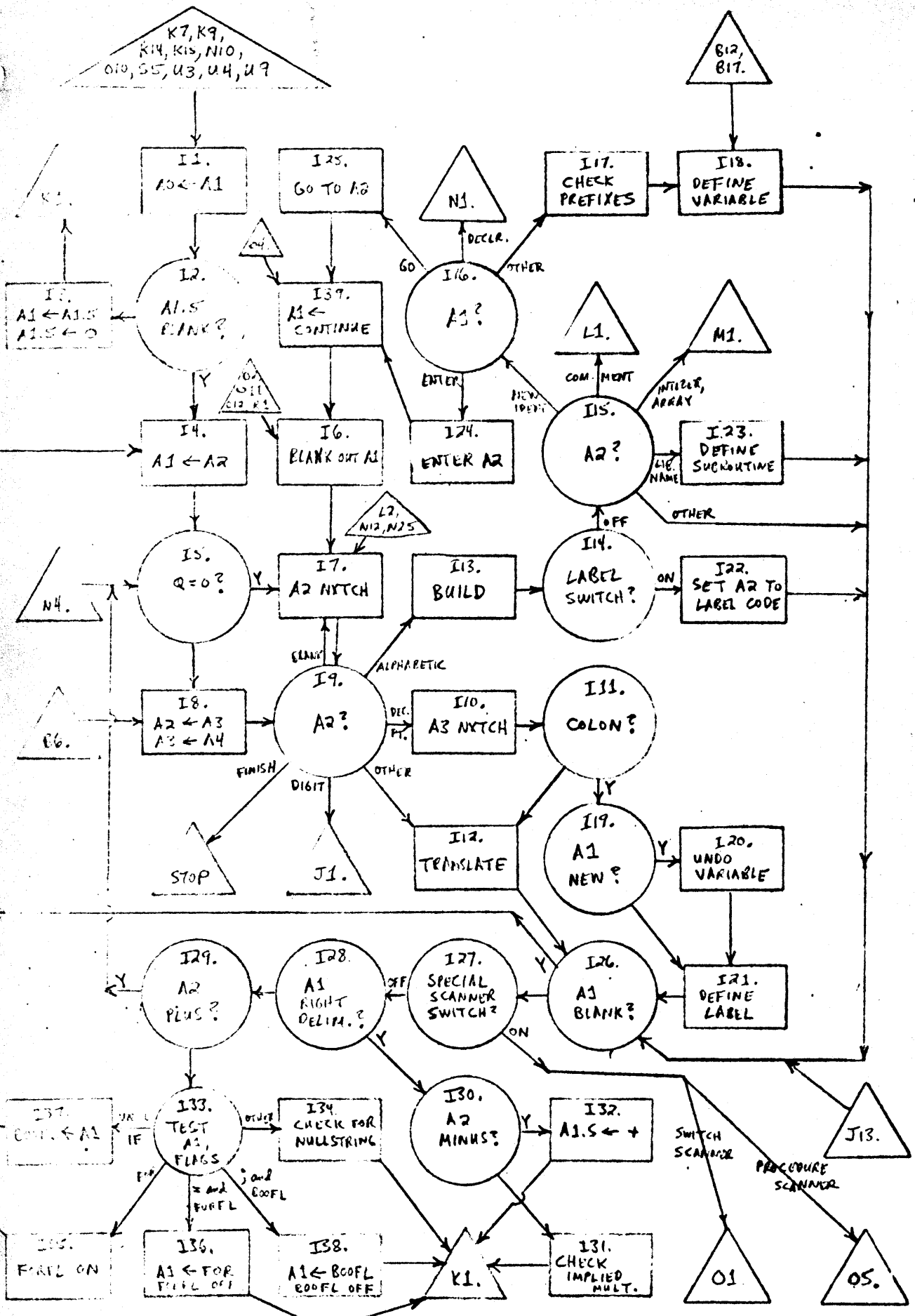
The general plan of attack in this compiler can be summarized as follows: The program looks at the input string language character by character from left to right. As soon as it sees something it can do, it does it; meanwhile it saves information about what the input has previously contained in counters and stacks. The control of this process is handled in the following way: There are five locations inside called A0,A1,A2,A3, and A4. Characters come in from the input string into either A2, A3, or A4 and from there they proceed leftward until they are processed. Most characters get all the way to A0 before they are processed, but some get no farther than A3. When characters originally come into A2, A3, or A4, they are in Cardatron code or in a special code for special characters. But the contents of A0 and A1 are in a completely different, more versatile compiler code. A2 is the transition where the codes are converted from one to another.

A "scanner" is a routine which looks at a string language by examining two (or more) adjacent characters at a time. There are four scanners in this compiler: the main arithmetic scanner (routine K) which operates from A0 and A1; the prescanner (routine I) which looks at A1, A2, A3, and A4; and the SWITCH and Procedure-call scanners (routine O) which operate from A1 and A2. The Rube Goldberg procedure call scanner actually begins by operation from A0 and A1 in conjunction with the arithmetic scanner, then shifts to A1-A2 after encountering a semicolon.

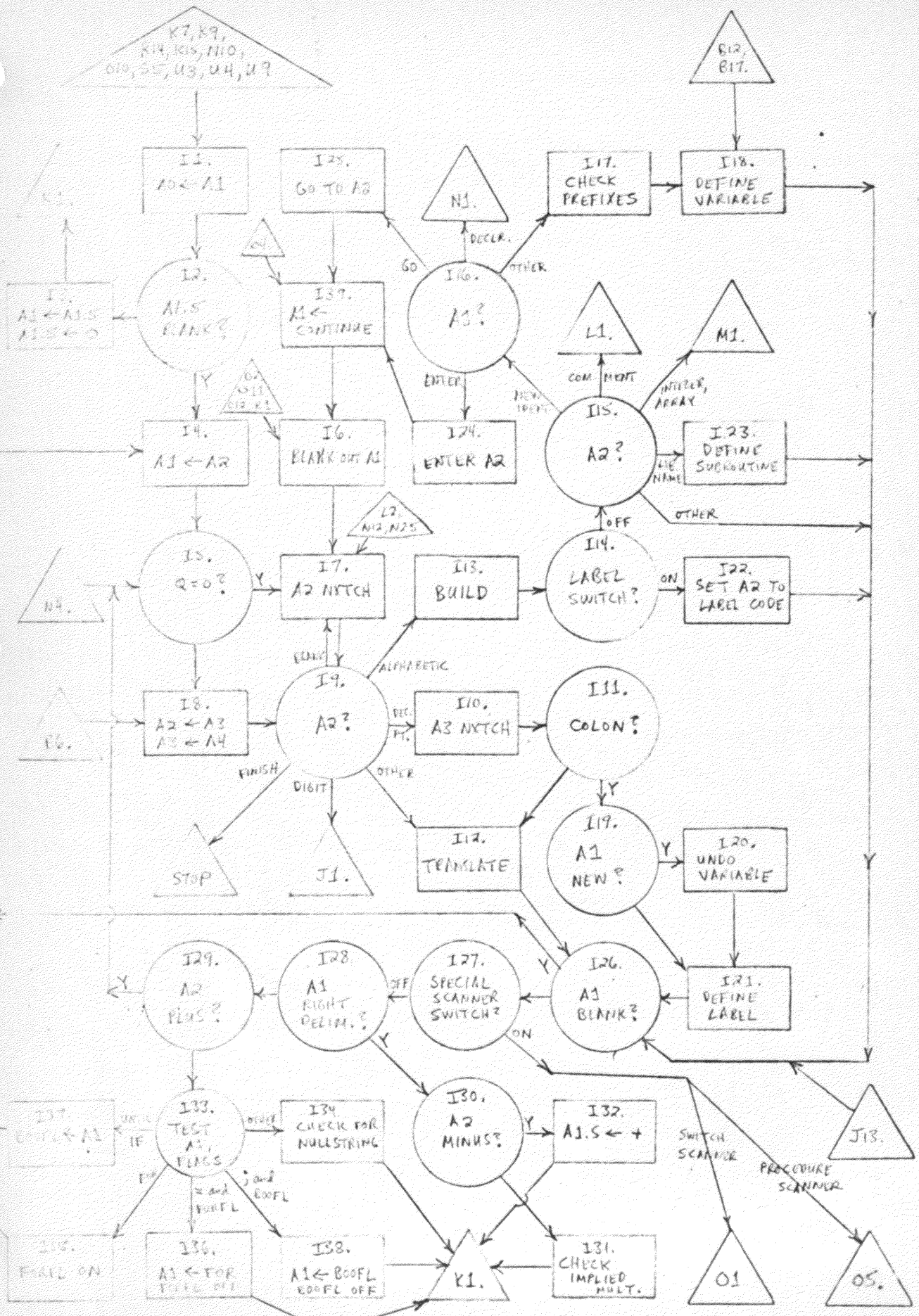
A "condenser" is a routine which takes a look at sequential characters in a string language and condenses them into a single entity or discards them entirely. There are six condensers in this compiler: the COMMENT condenser (routine L), the identifier-building condenser (subroutine B), the procedure input string condenser (part of routine N), and integer and array declaration condenser (routine M), the FORMAT condenser (part of routine N) and the constant condenser (routine J).

The PreScanner really seems to have the most control over the whole process, for it usually decides which scanner or condenser is to be used next. The job of the prescanner is not only to control the flow of the program, however; it also must edit the incoming language into a form which the other routines can digest. For example, it inserts multiplication sign when implied multiplication is indicated, it handles the translation of A2 from the original two-digit code to the compiler code, and it decides whether identifiers are labels or variables. Since the prescanner occasionally inserts a character there is an extra location called A1.5 which holds inserts.

Compilation begins with A0=a semicolon, A1, A1.5, and A2 blank. A counter Q tells whether A3 and A4 have any information; when Q=0 it means they don't, when Q=1 it means A3 does, and when Q=2 both A3 and A4 are filled. Q is initially zero. After the initialization, compilation begins in box I7.



I. PRESCANNER



I. PRESCANNER

I1. Move A1 into A0.
I2. Go to I4 if A1.5 is blank.
I3. Move A1.5 into A1, clear A1.5. To K1.
I4. Move A2 into A1.
I5. If Q greater than zero go to I8; else to I7.
I6. Blank out A1.
I7. NXTCH into A2 (Subroutine A with Q=0). To I9.
I8. Move A3 into A2, A4 into A3, and set Q=Q-1.
I9. If A1=FINISH, stop compilation (actually punch out the yet unpunched instructions and call in the library portion of the program). If A2 is blank, go to I7. If A2 is a number, go to J1. If A2 is a letter, go to I13. If A2 is a decimal point, go to I10. Else to I12.
I10. NXTCH inst A3 (Subroutine A with Q=1).
I11. If A3 is a second decimal point, go to I19. Else to I12.
I12. Change A2 into compiler code. To I26.
I13. Move A2 into A3 and BUILD an identifier (Subroutine B). Set A2 to the compiler code for this identifier.
I14. If the LABEL switch is on, go to I22 (this switch is normally off)
I15. If A2=COMMENT, go to L1. If A2 is a library function or procedure name that has not yet been used, go to I23. If A2 = INTEGER or ARRAY, go to M1. If A2 hasn't appeared before except possibly in an INTEGER declaration, go to I16. Else to I26.
I16. If A1= INPUT,OUTPUT,FORMAT,SUBROUTINE, or PROCEDURE, go to N1. If A1=ENTER, go to I24. If A1=GO, go to I25.
I17. In this case A2 is either a statement label or a variable. Let's acts under the assumption it's a variable. First we see if any prefixes for this label have occurred (Subroutine B, calling for special exit at steps B12,B17).
I18. Adjust arithmetic of the variable accordingly. If loop 6 is not full of simple variables, put the variable there, otherwise put it down on the drum. Go to I26.
I19. If A1 has not been coded as a variable, go to I21.
I20. Free up the location used for the variable and recode A1 as a label.
I21. Define A1 as a label equal to DRUML+LOOPL. To I26.
I22. Replace A2 by the label code, which is:
a) 5 CUB equiv for a regular label
b) 1 BUN equiv for a procedure parameter label
c) a simple variable code, for a defined procedure name
d) a procedure output variable code, for a procedure parameter procedure.
Go to I6.
I23. Reserve space for the library routine on the drum if needed. Calculate address of subroutine and enter it into the compiler code table. Go to I26.
I24. Compile ACALL to the label code for A2. (Subroutine E). TO I39.
I25. Compile the label code for A2 (Subroutine D). Go to I39.
I26. If A1 is blank, go to I4.
I27. At this point, A1 and A2 are both in compiler code. A switch is set normally to go to I28, or else to go to the SWITCH scanner 01, or to the Procedure-Call scanner 05.
I28. If A1 is a right parenthesis, END, constant or simple variable, go to I30.

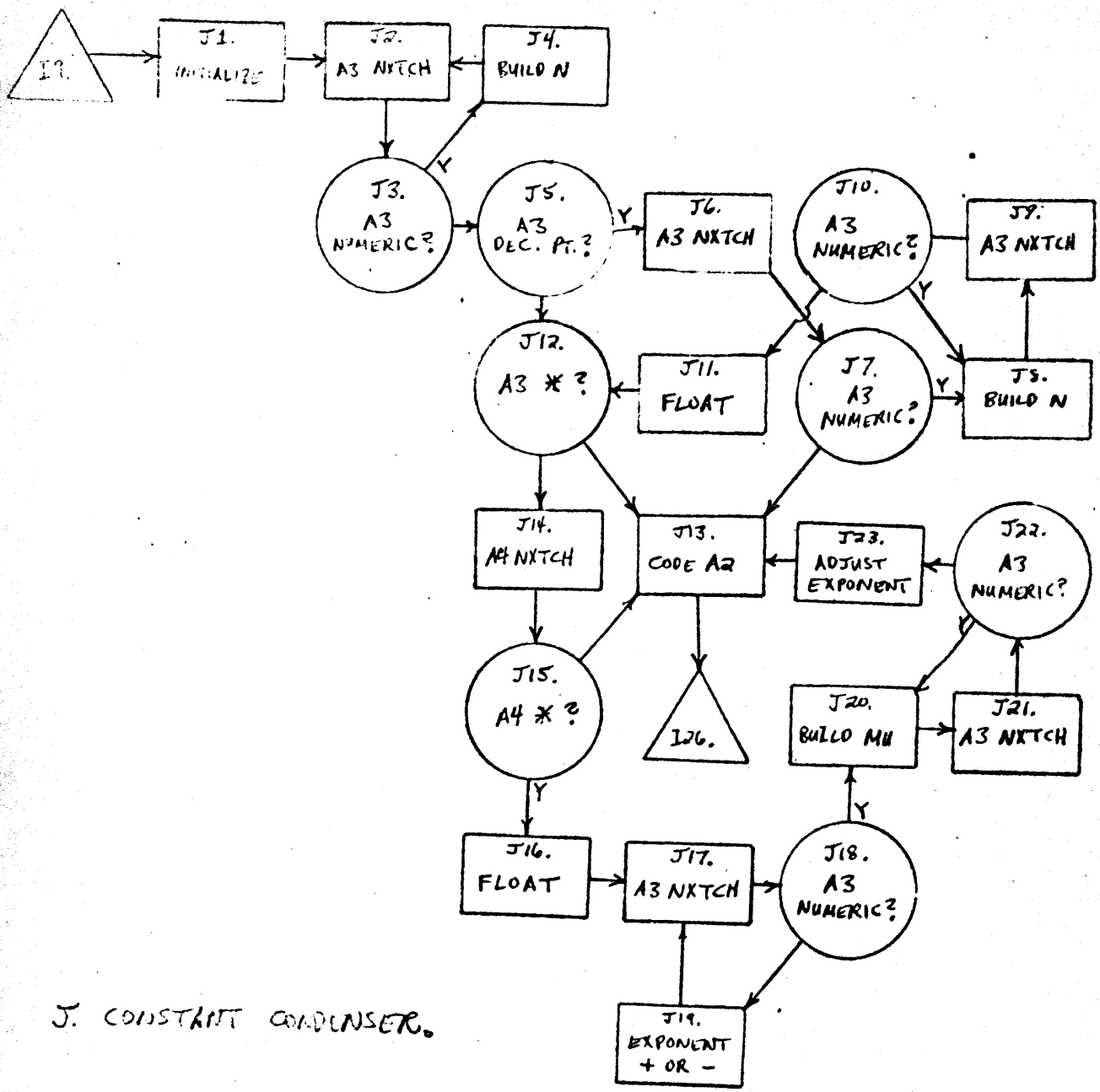
I29. If A2 is a plus sign, go to I5 (this ignores unary +); else go to I33.
I30. If A2 is a minus sign, go to I32 (this changes binary minus to unary minus).
I31. If A2 is a constant, any variable, function name, or procedure name, or a left parenthesis or BEGIN, set A1.5 to a multiplication sign. (This is the test for implied multiplication) Go to K1.
I32. Set A1.5 to plus sign. Go to K1.
I33. If A1=FOR, go to I35. If A1 is an equals sign and the FOR flag is on, go to I36. If A1=UNTIL or IF go to I37. If A1 is a semicolon and the Boolean flag is on, go to I38.
I34. If A1 is any of (left parenthesis, BEGIN, semicolon, STOP, comma) and A2 is any of (right parenthesis, END, semicolon, comma) set A1.5 to "nullstring" code. Go to K1.
I35. Set FOR flag on. Go to I4.
I36. Replace A1 by FOR, set FOR flag off, increment Doo counter.
I37. SEt Boolean flag to A1. Put DRUML+LOOPL with Boolean storage code on top of operand stack. To I4.
I38. Replace A1 by Boolean flag, turn this flag off. To K1.
I39. Set A1 = CONTINUE. To I5.

Coding Details: Temp Storage Used: A0,A1,A2,A3,A4,INSW,Q,PART2 table, EQUIV,LOOP6,HIGHL,SEG1-SEG5,TEMP1,FORFL,BOOFL,CNAME,CDOO,PROCX table. Subroutines Used: NXTCH,ACALL,OUTPT,LABLE.

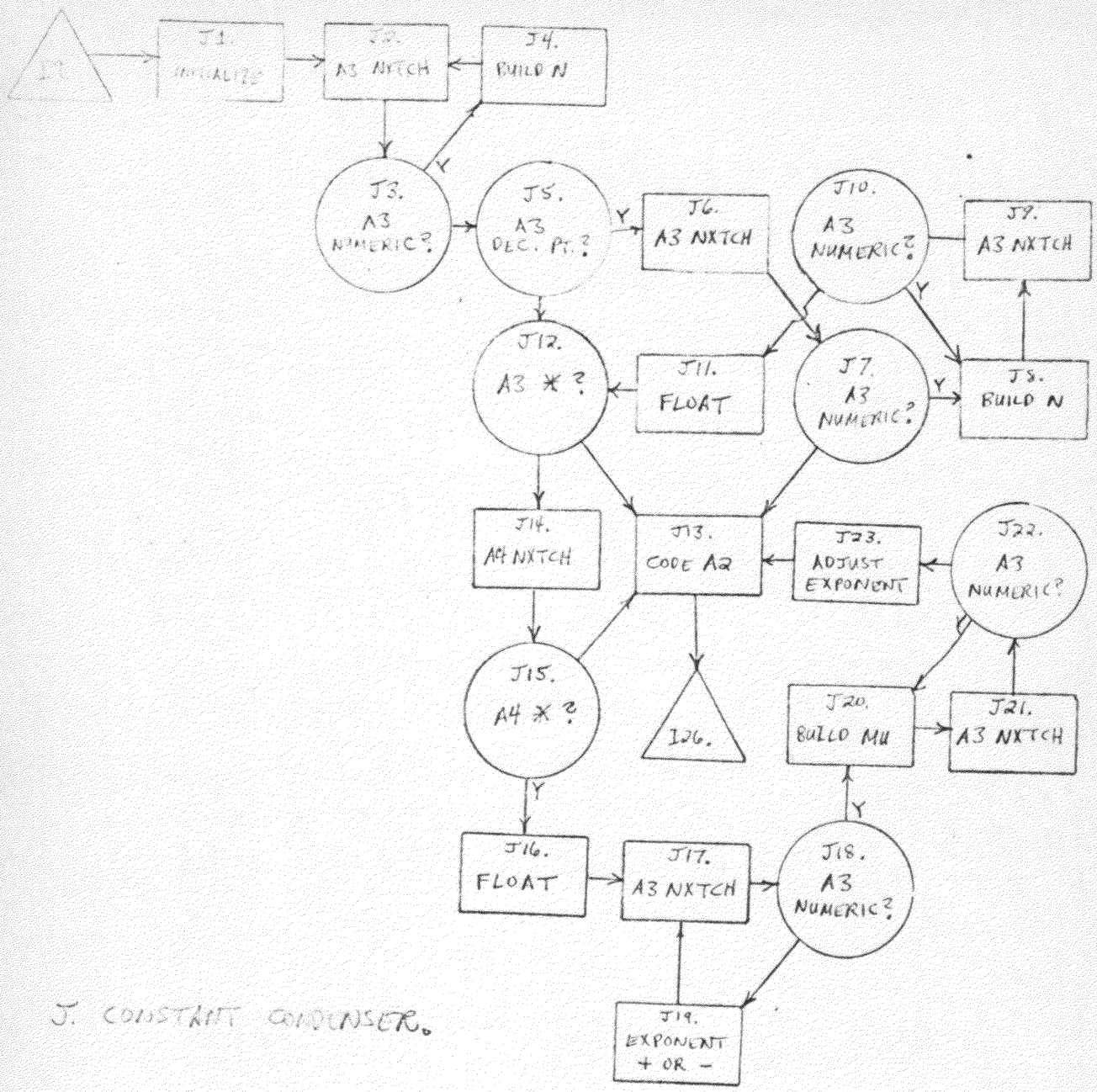
J. CONSTANT CONDENSER.

This routine changes constants from the input string code to 205 code. It is entered only from the Prescanner at a time when Q=0 and A2 is the first numeric character.

J1. Set N to A2, ARITH to fixed, MU=0.
J2. NXTCH into A3 (Subroutine A with Q=1).
J3. If A3 is not numeric go to J5.
J4. Set N to 10N+A3. To J2.
J5. If A3 is not a decimal point go to J12.
J6. NXTCH into A3 (Subroutine A with Q=1).
J7. If A3 is not numeric, the decimal point was a multiplication symbol which will be reinserted later because of implied multiplication. In this case go to J13.
J8. N = 10N+A3, MU=MU+1.
J9. NXTCH into A3 (Subroutine A with Q=1).
J10. If A3 is numeric, go to J8.
J11. Set ARITH floating. Convert N to floating point notation (divide by 10*MU).
J12. If A3 is an asterisk go to J14.
J13. Set A2 to constant code. To I26.
J14. NXTCH into A4 (Subroutine A with Q=2).
J15. If A4 is not an asterisk go to J13.
J16. If N hasn't yet been converted to floating point, set ARITH floating and convert N. Set Sign plus, MU zero.
J17. NXTCH into A3 again (Subroutine A with Q=1).
J18. If A3 is numeric go to J20.
J19. Set Sign plus or minus according to A3. To J17.
J20. MU=10MU+A3.
J21. NXTCH into A3.



J. CONSTANT CONDENSER.



J. CONSTANT CONDENSER.

J22. If A3 is numeric go to J20.
J23. Multiply N [illegible] by 10*±MU according to Sign. Tp J13.

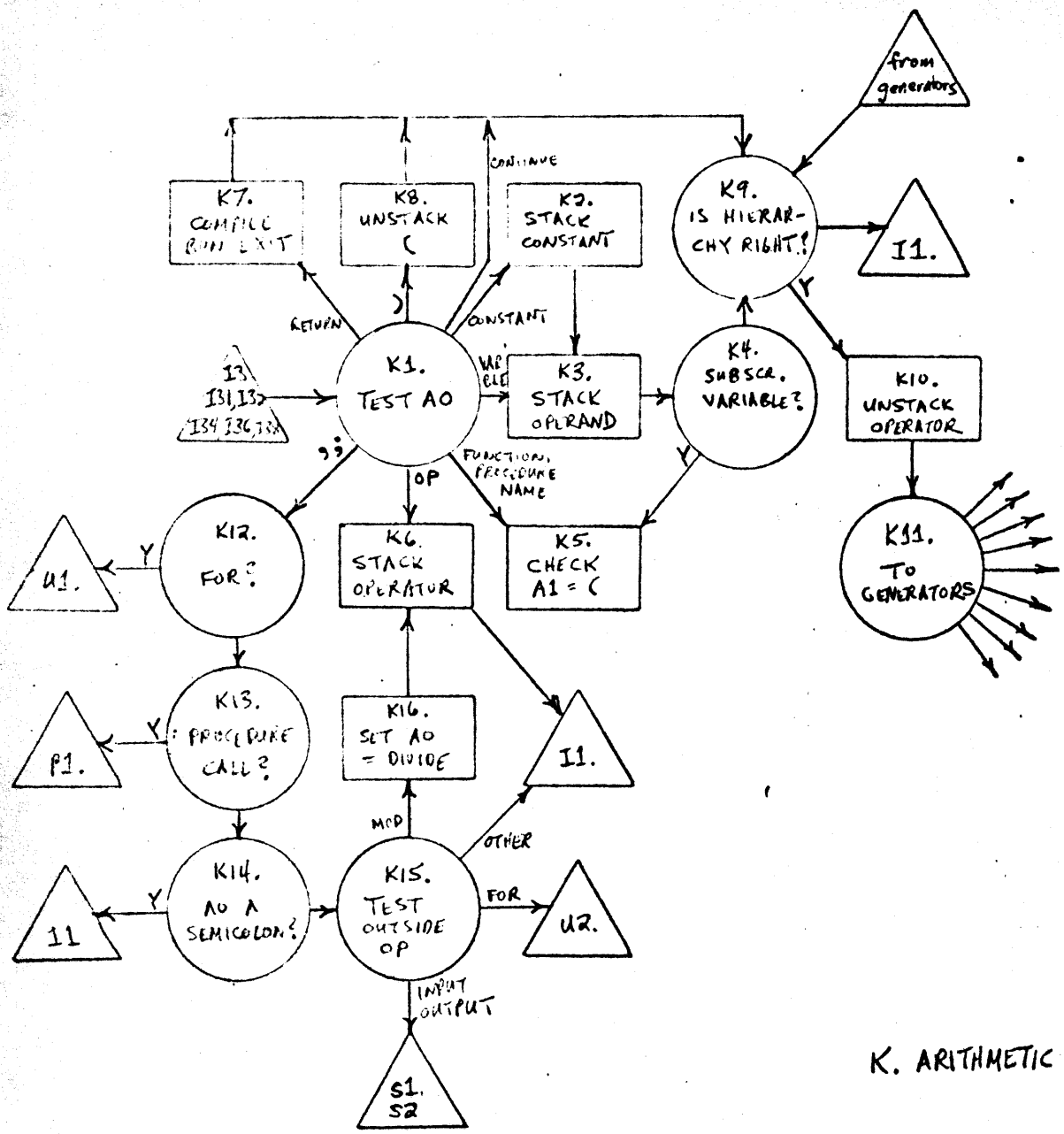
Coding Details: Temp Storage Used TEMP5,TEMP6,TEMP7,N
Subroutine Used ☎ NXTCH

K. ARITHMETIC SCANNER.

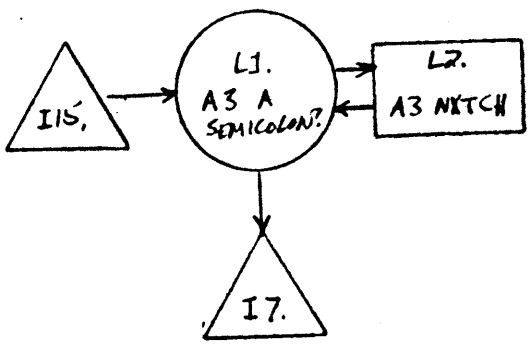
This consists mostly of tests on the type of quantity A0 is. The scanner has two main functions: a) to control what goes onto the operator and operand stacks; b) to detect when an operation is ready to be compiled for and in such an event to go to the proper generator. Between every two times the scanner is entered, the symbol pair A0,A1 has been moved to the right across the input language. The input language as seen by the scanner has been edited by the prescanner into a convenient form; a few non-arithmetic things come through here, but not many.

K1. If A0 is an operator, a left parenthesis or BEGIN, go to K6.
[illegible] If A0 is a semicolon or comma, go to K12. If A0=RETURN,go to K7.
If A0=CONTINUE, go to K9. If A0 is function or procedure name, go to K5. If A0 is a variable or "nullstring" go to K3.
If A0 is a constant, go to K2.
Otherwise,believe it or not, A0 is a right parenthesis or END. Go to K8.
K2. Put constant onto constant stack.
K3. Put A0 onto operand stack.
K4. If A0 is not a subscripted variable go to K9.
K5. ALARM if A1 is not a left parenthesis.
K6. Put A0 onto operator stack, go to I1.
K7. Compile BUN(exit) (Subroutine D, IN7). Go to K9.
K8. ALARM if top of operator stack does not pair with A0. Remove top of operator stack. Go to K9.
K9. If the heirarchy of the top of the operator stack is not greater than the heirarchy of A1, go to I1. (In this test we associate "heirarchies" with left and right parentheses, commas, and semicolons, as well as with all operators, but the "heirarchy" of a left parenthesis is tested only from the right, and that of a right parenthesis, [illegible] semicolon, or comma is tested only from the left.) The heirarchy table is:

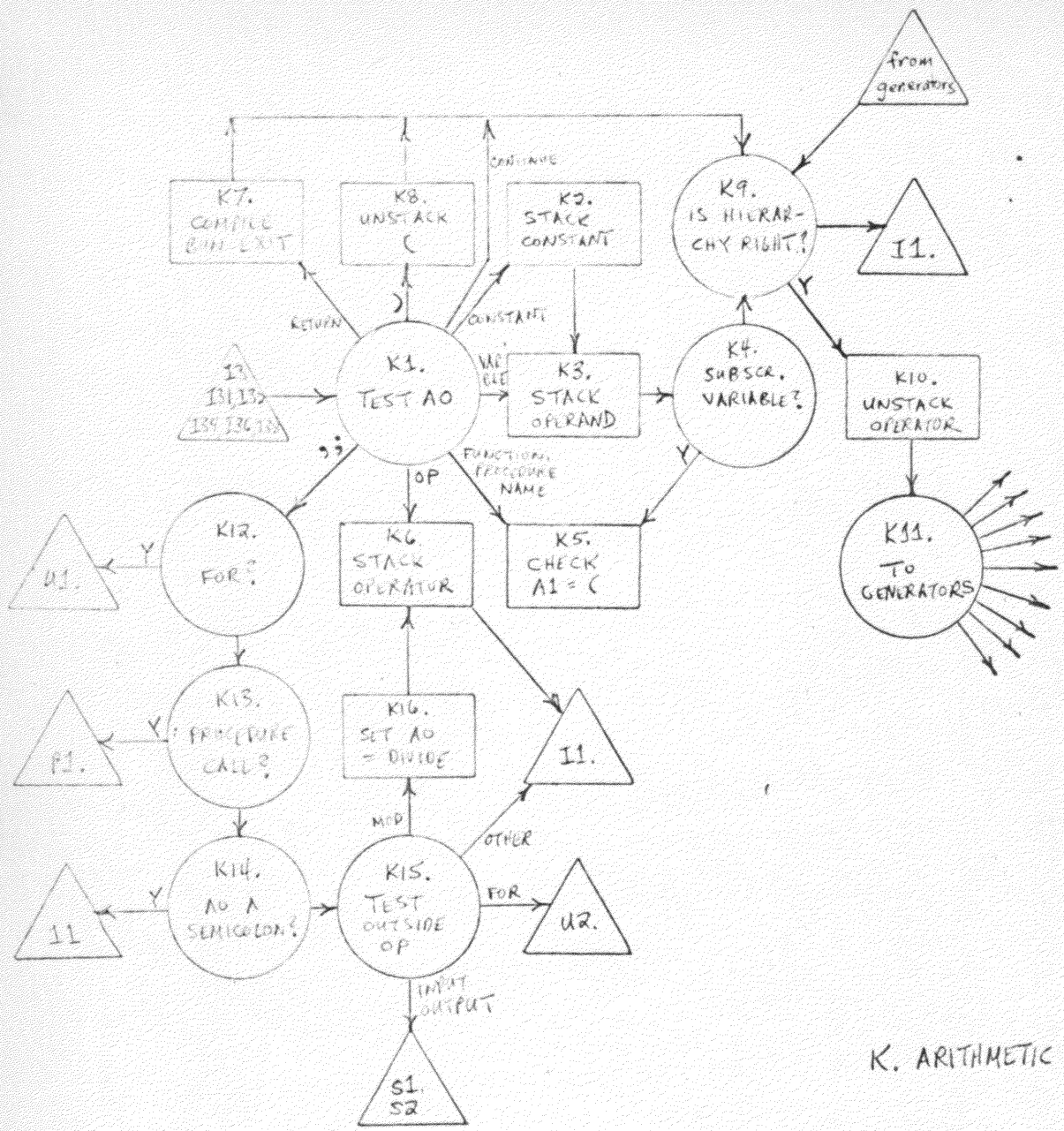
( ) BEGIN END FOR	00
[illegible] ;	02
UNTIL IF DO	04
STOP =	06
,	08
SWITCH	10
OR	12
AND	14
NOT	16
LSS,EQL,etc	18
","in MOD	19
+	20
/	22
·	24
- *	26
all others infinity	



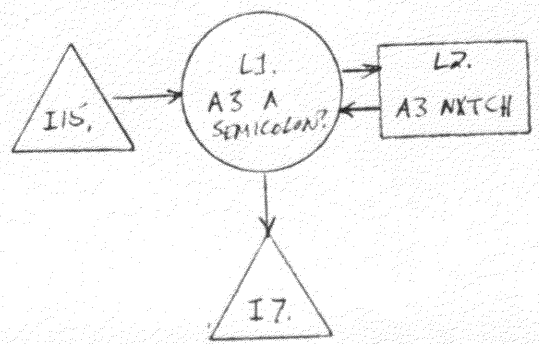
K. ARITHMETIC SCANNER.



L. COMMENT CONDENSER.

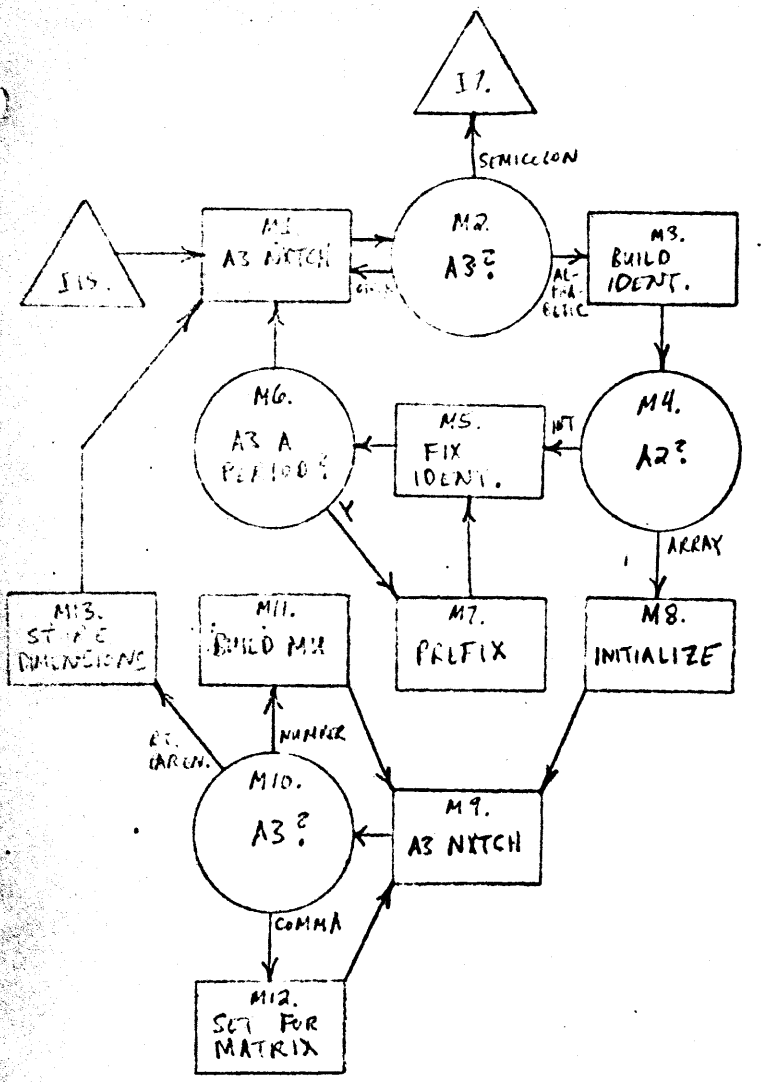


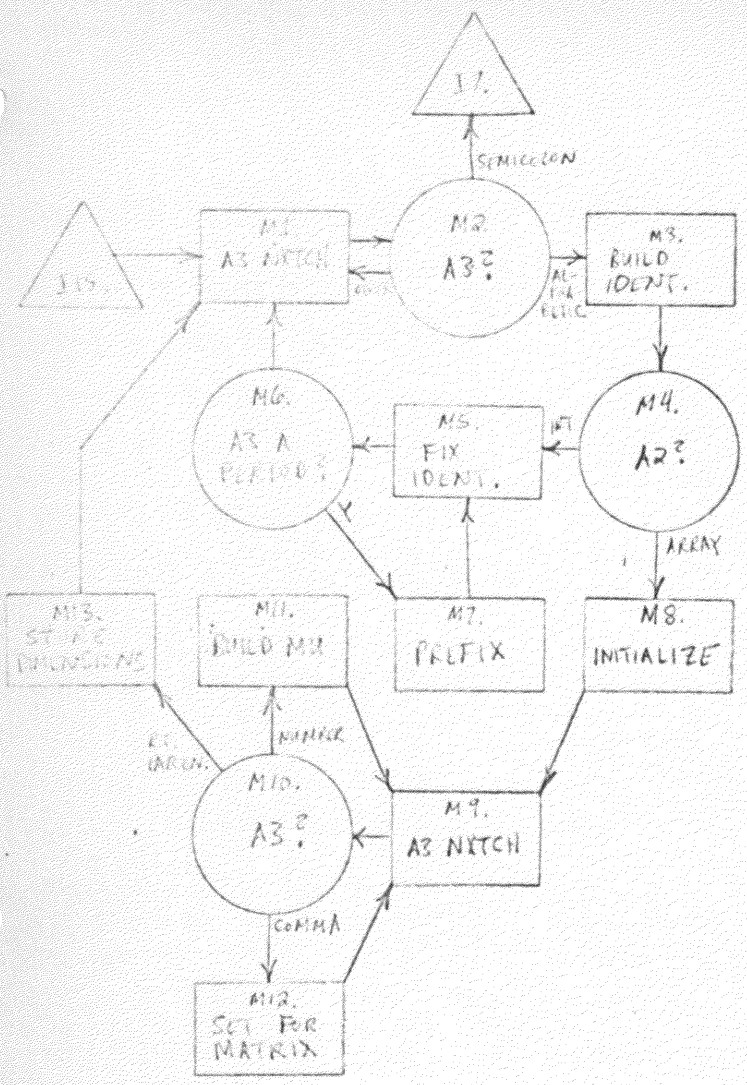
K. ARITHMETIC SCANNER.



L. COMMENT CONDENSER.

M. INTEGER, ARRAY DECLARATIONS.





M. INTEGER, ARRAY DECLARATIONS.

K10. Remove top of operator stack
K11. Go to generator for this operation. Generators are in sections P through U.
K12. If top of operator stack is FOR, go to U1.
K13. If next from top of operator stack is a procedure name, go to P1.
K14. If A0 is a semicolon, go to I1.
K15. If next from top of operator stack is INPUT or OUTPUT, go to S1 (OUTPUT) or S2(INPUT). If it is MOD, go to K16. If it is FOR, go to U2. Otherwise go to I1.
K16. Set A0 = divide with hierarchy 19. Go to K6.

Coding Details: Upon entry to the generators, rA=0,rB=CNAME-1, rR:88 = Operation code:08, and COUNT=0. These conditions were found to be most useful in the generators.
Temp Storage used: COP, OPST table, A0, CNO, NUMST table, CNAME, NSTAK table.
Subroutine Used: OUTP7.

L. COMMENT CONDENSER

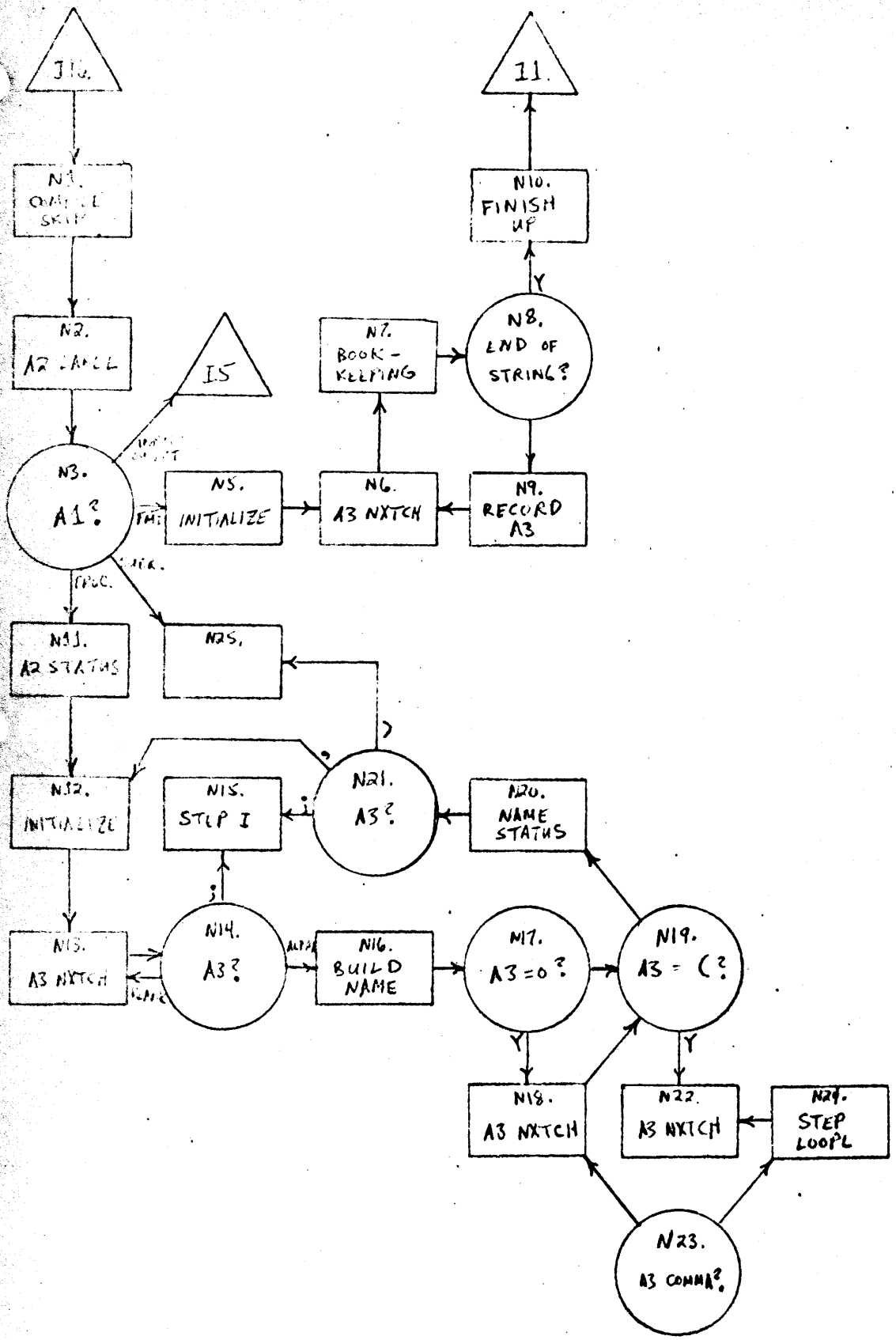
This routine is so trivial it requires no COMMENT.
L1. If Ao is a semicolon, go to I7.
L2. NXTCH into A3(Subroutine A with Q=1). Go to L1.
Coding Details, subroutines used NXTCH.

M. INTEGER AND ARRAY DECLARATION CONDENSER

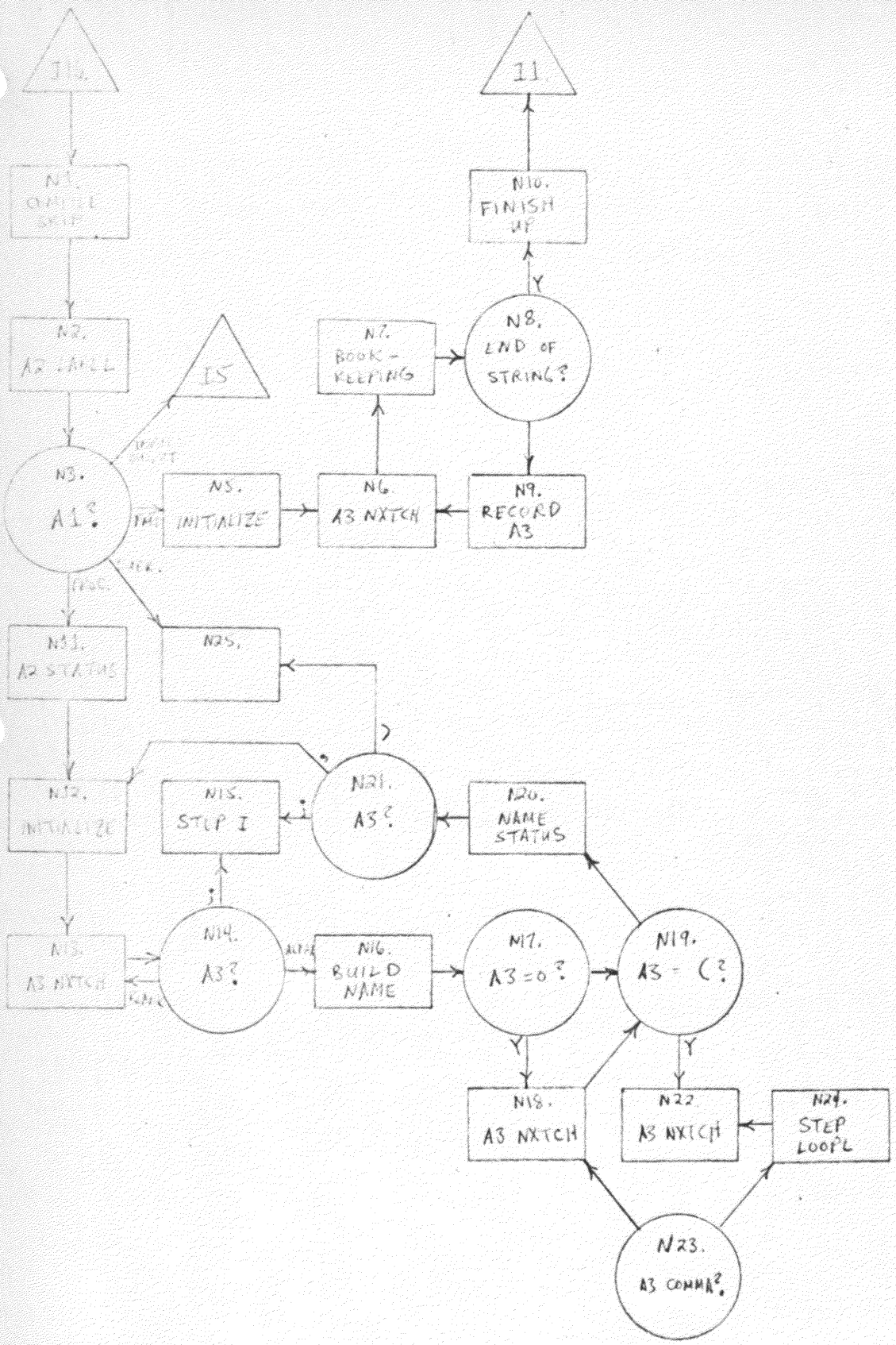
Entry is only from the Prescanner when A2=INTEGER or ARRAY, A3=blank,
Remarks: Prefixes apply only to those simple variables which appear only after the prefix.

M1. NXTCH into A3 (Subroutine A with Q=1).
M2. If A3 is a semicolon, go to I7. If A3 is alphabetic, go to M3. Else to M1.
M3. BUILD identifier (Subroutine B).
M4. If A2 is ARRAY, go to M8. Otherwise A2 is INTEGER.
M5. Make this name or prefix have integer meaning in compiler code.
M6. If A3 is not a period, go to M1.
M7. Set A3=blank and go to the prefix routine (Subroutine B;IN1).To M5.
M8. Set dimenxion to 1, MU=0.
M9. NXTCH into A3 ignoring all blanks (Subroutine A, with "last character" set to blank,Q=1).
M10. If A3 is numeric go to M11. If A3 is a comma, go to M12. Else to M13.
M11. MU = 10MU + A3. Go to M9.
M12. Set dimension to 2, NU=MU,MU=0. Go to M9.
M13. If dimension is one, reserve MU locations. If dimension is two, reserve MU*NU locations. Set compiler code for this name with proper base address and dimensionality, retaining INTEGER status if previously declared. Go to M1.

Coding Details: Temp Storage Used A3, TEMP5, TEMP6, TEMP7, HIGHL, PART2 table, LSTCH.
Subroutines Used: NXTCH, BUILD, BLD1.



N. DECLARATIONS.



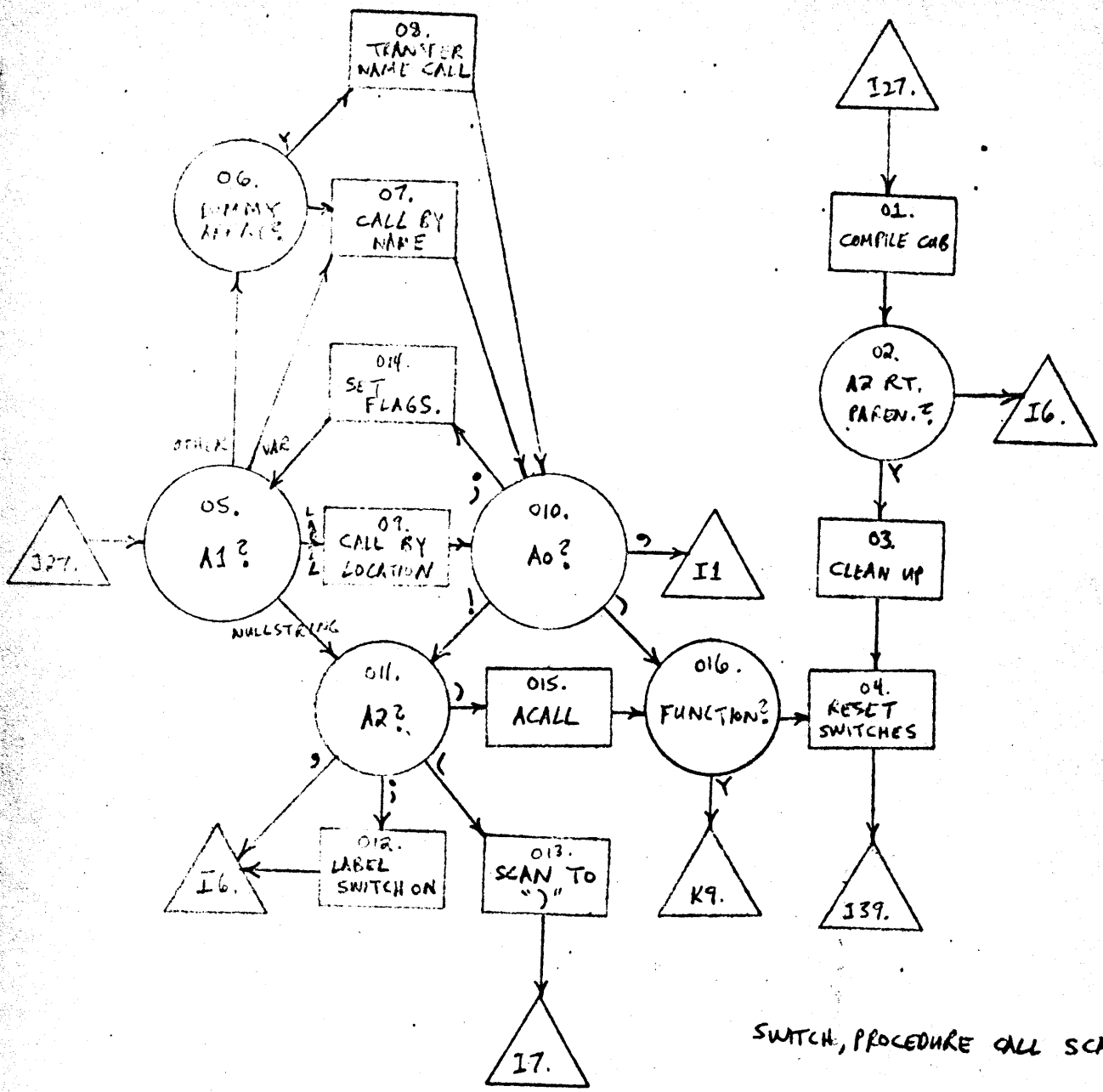
N. DECLARATIONS.

N. DECLARATIONS.

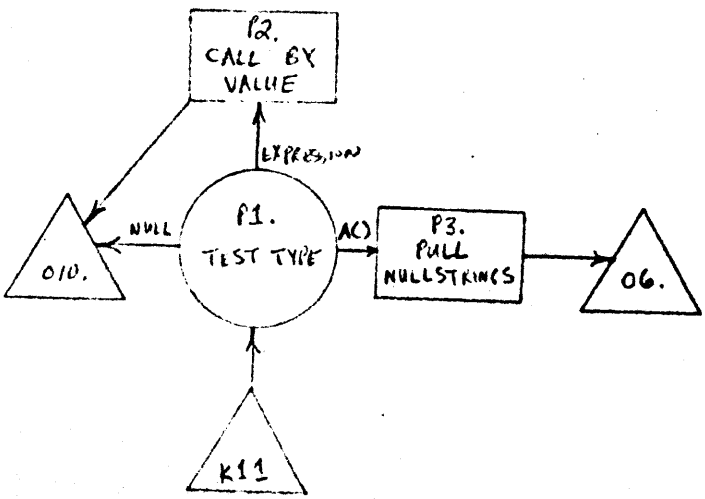
This routine processes the "beginning phase" of INPUT, OUTPUT, FORMAT, SUBROUTINE, and PROCEDURE declarations. The "middle phase" is handled by the normal scanner. The "ending phase" of these declarations is done by generators. N5-N10 is the FORMAT condenser, N12-N24 is the procedure input string condenser.

N1. Compile CUB(around). (Subroutine D, entry IN7). This is done so that the coding produced by the declaration is bypassed at the time it appears in the running program.
N2. Define A2 to be a label located at DRUML.
N3. If A1 is INPUT or OUTPUT, go to I5. If it is SUBROUTINE, to N25. If A1 is FORMAT, go to N5. Otherwise A1 is PROCEDURE, go to N11.
N4. undefined
N5. Set A2= left parenthesis, set STAR=0, FWORD blank.
N6. NXTCH into A3, bringing in actual untranslated alphanumerics if we are between two asterisks, ignoring all blanks if we are outside of asterisks. (Subroutine A with special flags)
N7. If A3 is an asterisk, set STAR = 1 - STAR.
N8. If A3 is a right parenthesis, and STAR = 0, go to N10.
N9. Add A3 to FWORD. If FWORD is full, punch it (Subroutine C) and set it to blanks. Go to N6.
N10. Punch FWORD (Subroutine C). Go to I1.
N11. Set temp. storage counter so as not to overlay any which occurred previously. Define A2 to be a procedure label with global significance. Set up procedure heading.
N12. Set I = 0, J = 0.
N13. NXTCH into A3 (Subroutine A with Q=1).
N14. If A3 is a semicolon go to N15. If alphabetic, go to N16. Otherwise A3 ought to be blank; go to N13.
N15. Set I=I+3,J=I. Go to N13.
N16. BUILD an identifier (subroutine B).
N17. If A3 ≠ 0, go to N19.
N18. Put next nonzero character into A3 (Subroutine A,Q=1)
N19. If A3 is a left parenthesis, go to N22. ALARM if A3 is alphabetic or numeric.
N20. Step LOOPL by one. Set the identifier last encountered to have the meaning specified by J: (simple variable, proc. param. vector, proc. param. matrix, proc. param. output, proc. param, vector, proc. param. matrix, proc. param. label, proc. param. procedure) respectively for J= 1 through 8. Then set J = I.
N21. If A3 is a semicolon go to N15. If A3 is a comma, go to N13. Otherwise, A3 is a right parenthesis; go to N25.
N22. J=J+1. Set A3 = next non-blank character (Subr. A, Q=1, last char=0).
N23. If A3 is a comma, go to N24. Otherwise A3 is a right parenthesis; go to N18.
N24. Step LOOPL by one. To N22.
N25. Compile STA exit (for either procedure or subroutine)(Subroutine D). Then go to I7.

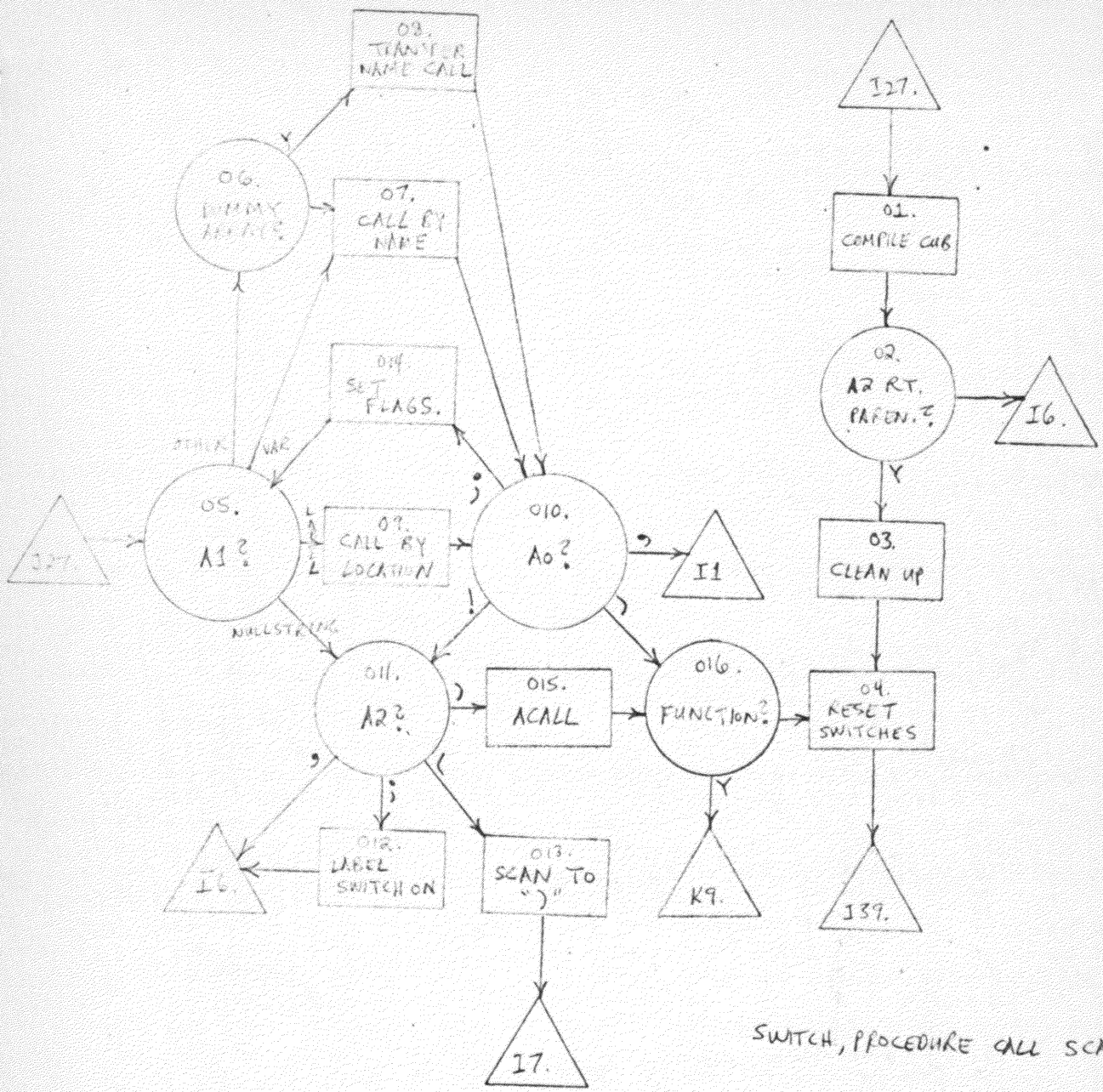
Coding Details: Temp Storage Used: A1,A2,PART2 table, DRUML,LOOPL,TEMP5, TEMP6,TEMP7,THISE,PROCC,PROCT.
Subroutines used: NXTCH,DFINE,PUNCH,OUTPT.



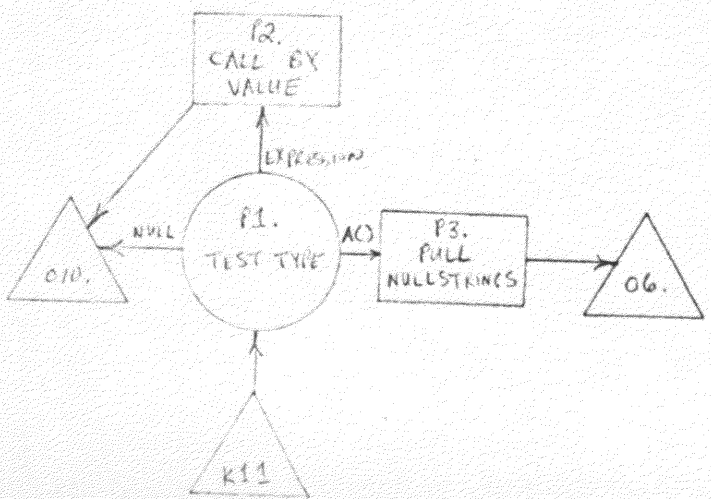
SWITCH, PROCEDURE CALL SCANNERS.



P. PROCEDURE GENERATOR.



SWITCH, PROCEDURE CALL SCANNERS.



P. PROCEDURE GENERATOR.

O. SWITCH, PROCEDURE CALL SCANNER.

This scanner alters the normal mode of compilation wildly. The SWITCH and PROCEDURE generators do the preliminary work before this routine takes over. Remarks: The entrance to the procedure call scanner in its first phase (from the Scanner) is made to the generator P1; the second phase enters at 05 from the Prescanner. The switch scanner enters at 01 from the Prescanner.

01. Set DRUML=TEMP7=TEMP7+1. Compile label code for A1 (Subr. D).
02. If A2 is a right parenthesis, go to 03. Else to I6.
03. Fix up forward reference made by SWITCH generator.
04. Reset I27 switch and label switch to normal position. Adjust A0 so it will be bypassed at next entrance of scanner. To I39.
05. If A1=nullstring, go to 011. If it is a label code for a non-parameter label, go to 09. If it is a simple variable, go to 07. Otherwise go to 06 (testing A1).
06. If it is a procedure parameter array, go to 08.
07. Here we have an ordinary call by name which is to be entered as input to a procedure. Compile CAD (const) (Subroutines D,F), STA (procedure input storage) . For a matrix, do this twice. To 010.
08. Compile CAD (current procedure input storage), STA (called procedure input storage)(Subroutine D). For a matrix, do this twice. To 010.
09. Provide for four locations in loop7 buffer (Subroutine F,IN1), Compile CAD (self plus 2), BUN (self plus 2), label code, STA (called procedure input), all four of these instructions(Subr. D).
010. If A0 is a right parenthesis, go to 016. If it is a comma, go to I1. If it is a semicolon go to 014. Otherwise A0 is an exclamation mark.
011. If A2 is a right parenthesis, go to 016. If it is a left parenthesis,go to 013. If it is a semicolon, go to 012. Otherwise it's a comma and we go to I6.
012. Set label switch on, go to I6.
013. NXTCH into A3 until it is a right parenthesis (Subroutine A with Q=1). Then set A1=nullstring. To I7.
014. Set A0=exclamation mark. Set I27 switch to jump to 05. Go to 05.
015. Compile to ACALL the procedure. (Subroutine E).
016. If A0 is a right parenthesis, we are using the procedure as a function; go to K9. Otherwise A0 is an exclamation point and we go to 04.

Coding Details: Temp Storage Used: OPST table, CNAME, CMTX, A0, A1, A2, TEMP1, TEMP6, TEMP7, PROSW,SWTCH,DRUML.
Subroutines Used: NXTCH, CONOT, OUTPT, PRSUB, FULL, GET, LABLE, ACALL.

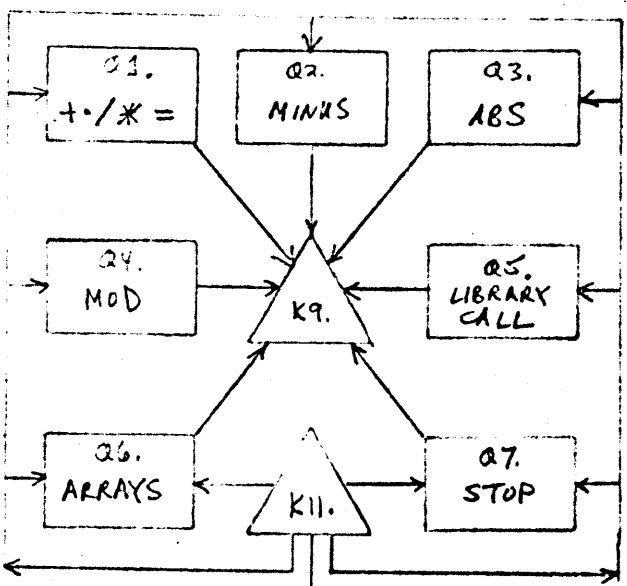
GENERATORS

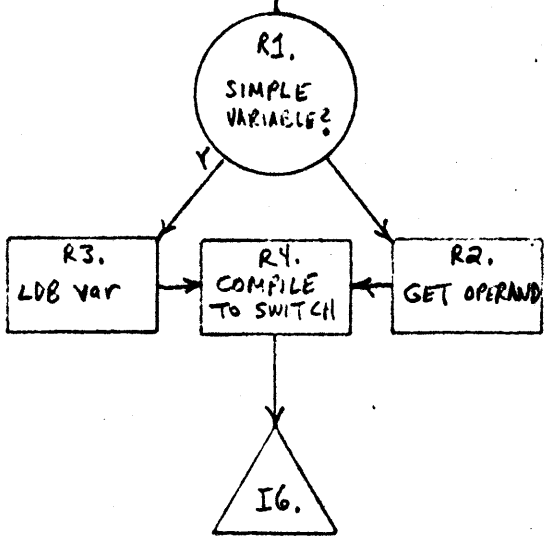
The remaining routines are the special processors which handle their own little feature of the language.

P. Procedure generator. Used for library procedure calls and previously defined procedure calls.

P1. If the top of the name stack is the nullstring code go to 010. If it is an array and the top of the subscript stack is a nullstring, go to P3.

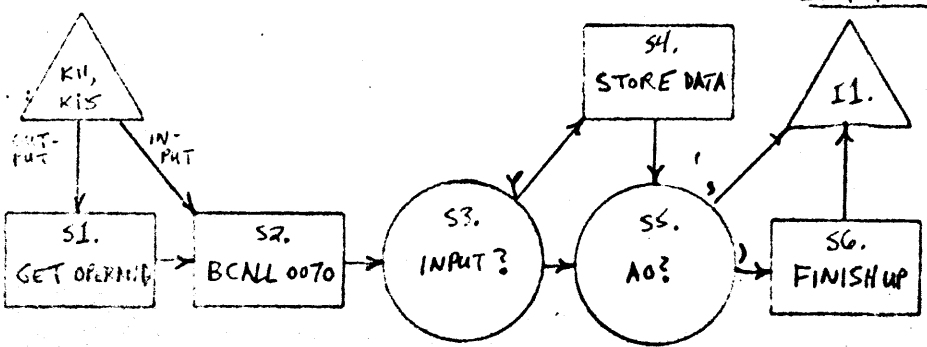
Q. ARITHMETIC GENERATORS.



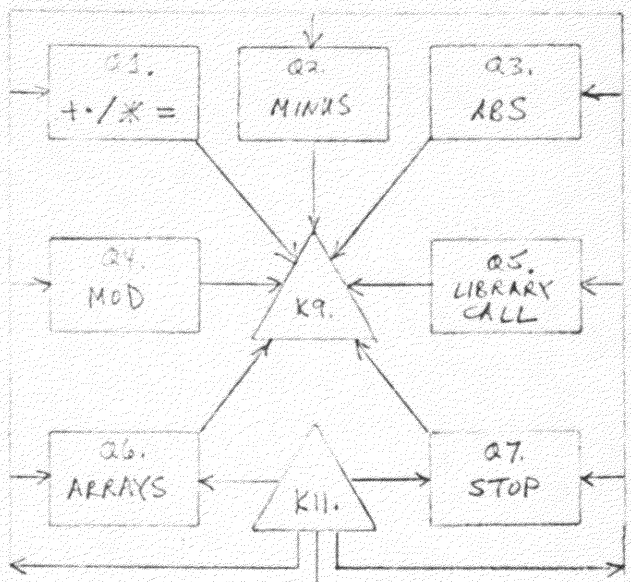


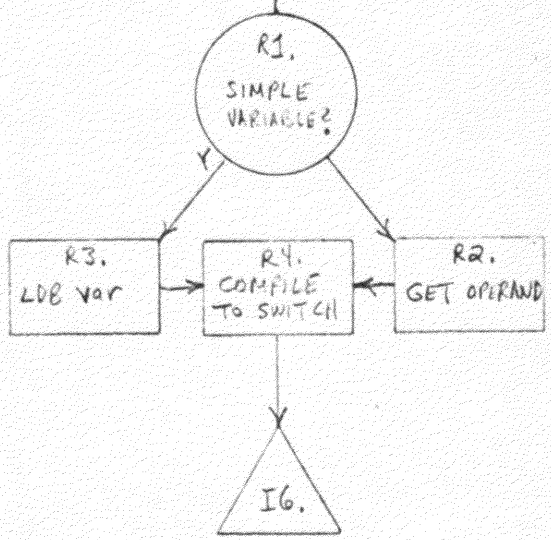
R. SWITCH GENERATOR.

S. INPUT-OUTPUT GENERATORS.



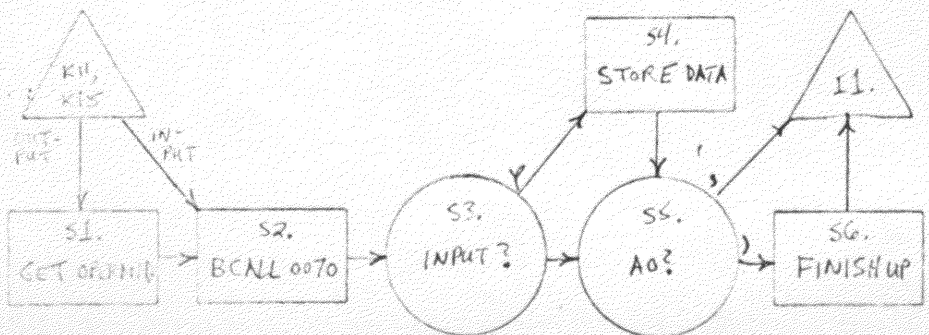
Q. ARITHMETIC GENERATORS.





R. SWITCH GENERATOR.

S. INPUT-OUTPUT GENERATORS.



P2. GET it (Subroutine G) and compile STA (procedure input storage). To 010.
P3. Remove all "nullstringg" from subscript stack, go to 06 (testing top of name stack).

Q. Arithmetic Generators. Addition, Multiplication, Division, Exponentiation, Replacement go to Q1. Minus goes to Q2. ABS goes to Q3. MOD goes to Q4. Library functions go to Q5. Arrays go to Q6. "STOP" goes to Q7.

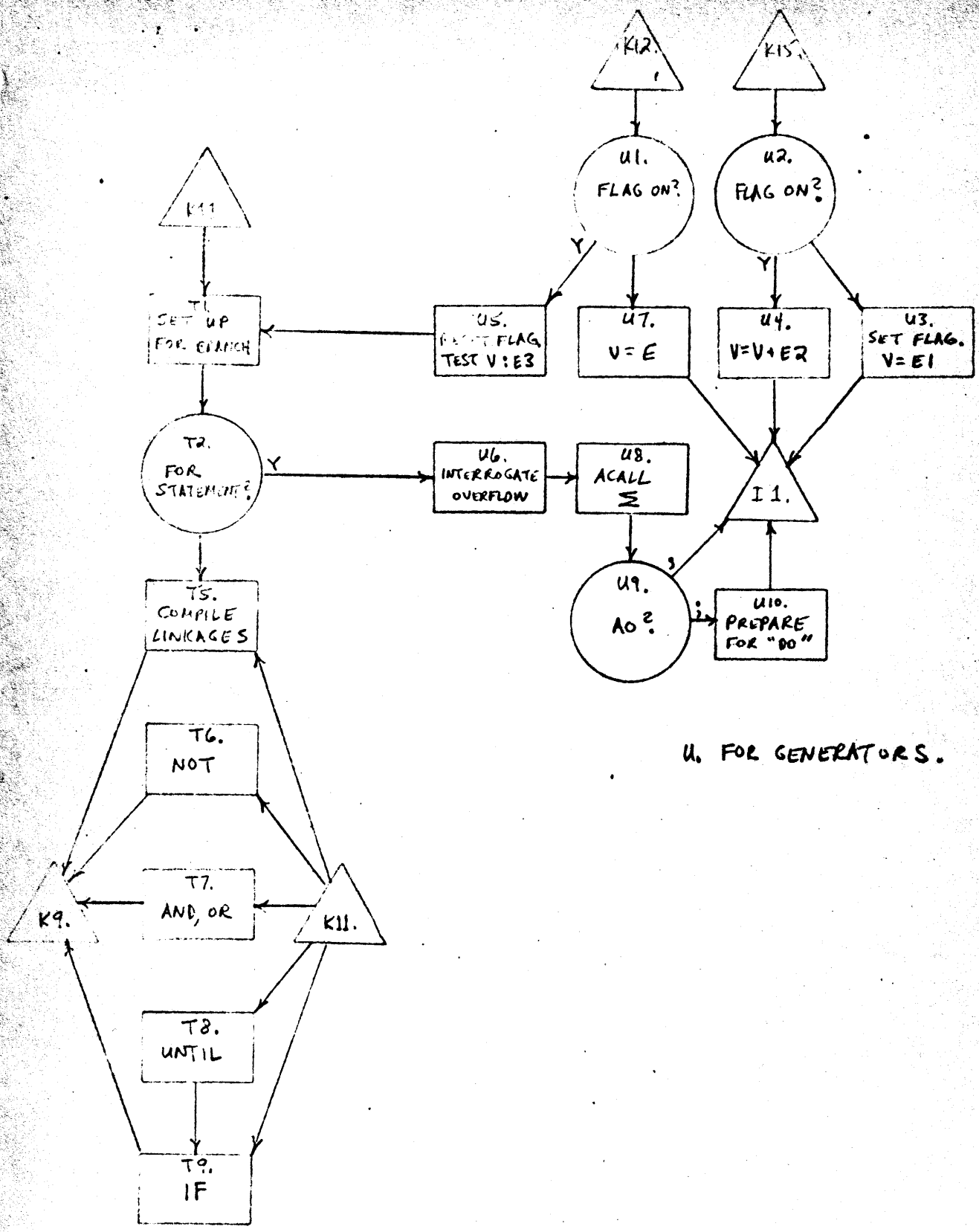
Q1. If we have a simple variable or a temp storage location raised to the second power, change it to a multiply. Compile for the proper operation (Subroutine H, entry IN1). To K9.
Q2. Reverse sign of top of operand stack. To K9.
Q3. Set absolute value tag of top of operand stack on, set it positive. To K9.
Q4. Compile SLT 10 (Subroutine D). Go to K9.
Q5. GET top of operand stack (ALARM if fixed) (Subroutine G). BCALL the library function (Subroutine E). To K9.
Q6. Move subscripts of operand stack to subscript stack. If they are arrays, GET them first (Subroutine G) and replace them with temp. storage symbols. Take care not to introduce any holes in the subscript or constant stacks! Go to K9.
Q7. If top of operand stack is not a nullstring, GET it (Subroutine G). Remove it from stack. Compile STOP 0137 (Subroutine D). To K9.

R. Switch Generator. When R1 is entered the hierarchy has been rigged so that A1 is a comma, A2 a left parenthesis.
R1. If top of operand stack is a fixed point simple variable, go to R3.
R2. GET top of operand stack (Subroutine G). If it is floating, compile FAD (5810000000) (Subroutines D,F). Compile STA 4001, LDB 4001, (Subroutine D) and go to R4.
R3. Compile LDB (variable) (Subroutine D).
R4. Compile -BUN (next) (Subroutine D, IN7). Set TEMP7= DRUML; compile CUB (fwd. ref) (Subr. D again). Set I27 switch to exit to O1. To I6.

S. Input-Output Generators. S1 is OUTPUT entry, S2 the INPUT entry.

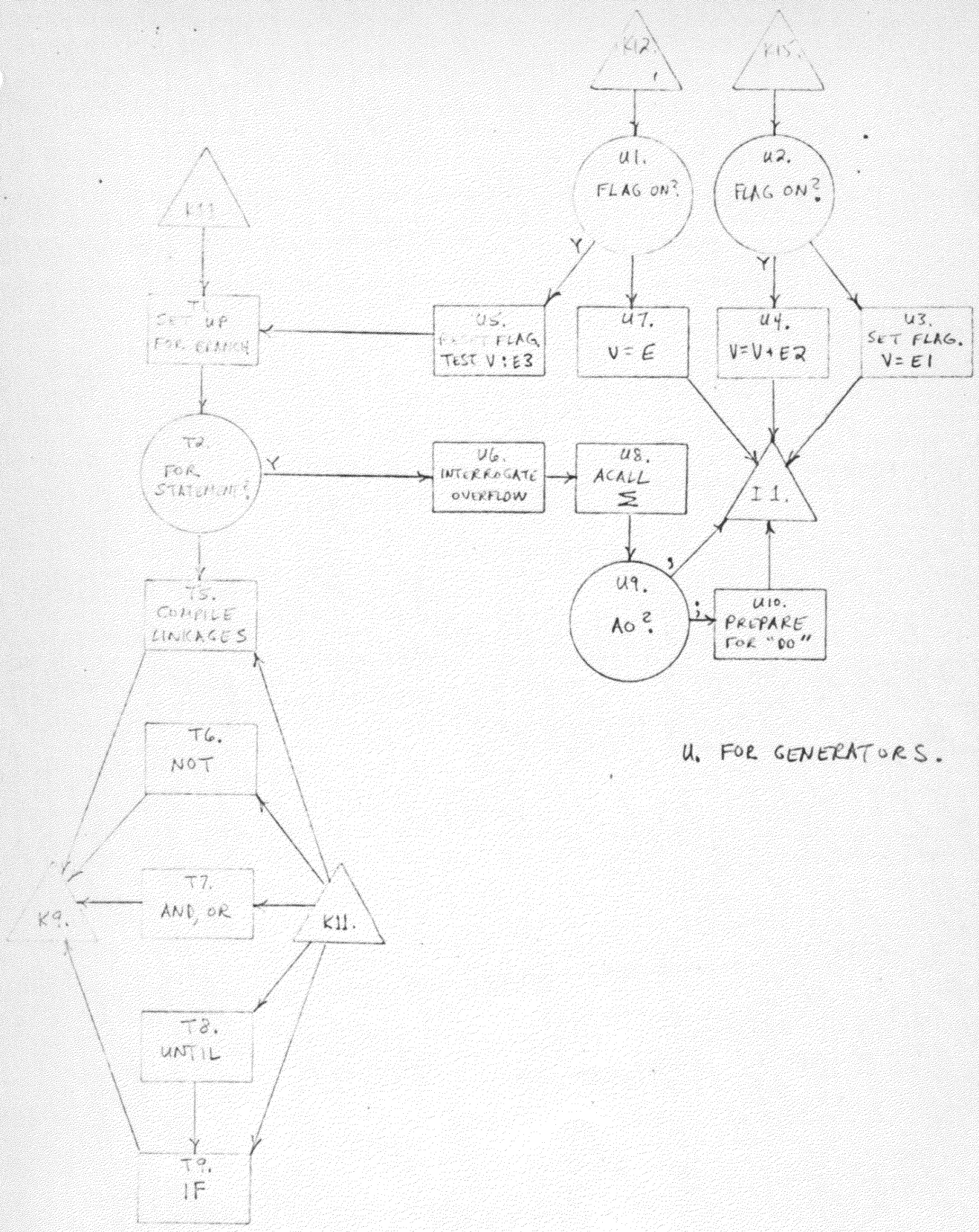
S1. GET top of operand stack (Subroutine G).
S2. BCALL the coroutine line (Subroutine E). (i.e., BCALL location 0070)
S3. If INPUT, go to S4; if OUTPUT, to S5.
S4. Set RIGHT= temp storage code with same type (fixed or floating) as top of operand stack. Compile to store (Subroutine H, OP=replace).
S5. If A0 is a comma, go to I1; else it is a right parenthesis.
S6. Compile CAD self, CIRA 4 BUN to the coroutine line (Subroutine E). Go to I1.

T. Relation Generators. PCS enters at T5. EQL,GTR,etc enter at T1. NOT enters at T6, AND-OR at T7, UNTIL at T8, IF at T9. IF and UNTIL generators take over after the statement following the relations has already been processed.



U. FOR GENERATORS.

T. RELATION GENERATORS.



U. FOR GENERATORS.

T. RELATION GENERATORS.

T1. Negate the sign of the top operand and compile to subtract the two quantities (Subroutine H, OP=non commutative addition). If we have EQL or NEQ, compile CNZ (self plus 2), otherwise if the last instruction was ADD,SUB,FAD,or FSU some constant, we alter this constant to get the appropriate branching situation, in the other event compiling ADD (+9999999999) (Subroutines D,F).
T2. [illegible] IF we are in the FOR routine, go to U6. Else to T5.
T3. undefined
T4. undefined
T5. Compile either 9CUB 0000 (for PCS) or 1CUB 0000 (for EQL,NEQ) or 1CCB 0000 (for GTR,GEQ,LSS,LEQ) (Subroutine D, IN7). Then compile CUB 0000. The 0000's will be filled in later. Record the location of these 0000's, and whether they represent a true or false situation, in the true-false stack. Go to K9.
T6. Change all the true-false conditions of the topmost relation entries in the true-false stack. Go to K9.
T7. Combine the two top relations of the true-false stack into a single one; fix up forward references for those of the first relation feeding them into the second relation where it has to be tested in order to obtain the proper AND-OR condition. To K9.
T8. Compile CUB (back to relation test). (Subroutine D, IN7).
T9. Fix up remaining forward references for the top relation in the true-false stack, and pull it off the stack. Some of the forward references go to the beginning of the statement, others bypass it. Then go back to K9.

U. FOR generators. Entrance is from the scanner when A0 is "," or ";".

U1. If special-comma flag is on go to U5, otherwise to U7.
U2. We have the (E1,E2,E3) case where we're scanning one of the two commas. If the special flag is on, go to U4.
U3. Turn special-comma flag on. Look and see if A1 is a minus sign or not, and remember this. Compile to set the variable equal to E1 (Subroutine H, OP=replace). Compile CUB (test E1 vs. E3). Go to I1.
U4. Compile to add E2 to the variable (Subroutine H). Compile to store the result in the variable (Subroutine H again). To I1.
U5. Turn off special-comma flag. Go to GEQ or LEQ (depending on whether E2 began with minus sign) generator, T1.
U6. Compile either BOF (self plus 2) CUB back, or CCB back depending on branching situation. To U8.
U7. Compile to set the variable to this value (Subroutine H,OP=replace).
U8. ACALL the next statement. (Subroutine E).
U9. If A0 is a comma go to I1. Else A0 is a semicolon.
U10. Compile CUB around (Subroutine D,IN7). Change operation from FOR to DO,and compile STA exit as in a subroutine. Go to I1.

..... and that's all there is to this compiler.