U. S. NAVAL WEAPONS LABORATORY

TECHNICAL MEMORANDUM

4 December 1959

No. K-32/59

A DESCRIPTION OF

AN ALPHANUMERIC COMPILER FOR NORC

Mary Louise Hagemeyer

Computation and Analysis Laboratory

REVISED 25 AUGUST 1960

Approved by:

Ralph A. Niemann

Ralph A. Niemann
Director, Computation
and Analysis Laboratory

This memorandum is not to be construed as expressing the opinion of the Naval Weapons Laboratory, and while its contents are considered correct, they are subject to modification upon further study.

Copies may be obtained from the Director, Computation and Analysis Laboratory.

TABLE OF CONTENTS

NWL Technical Memorandum No. K-32/59

A DESCRIPTION OF

AN ALPHANUMERIC COMPILER FOR NORC

Distribution:
K, K-1
KB, KBP, KBX
KC, KCD, KCM, KCO, KCP, KCR, KCX
ACL

Abstract

The acquisition of a 20,000-word memory, together with the development of the Universal Data Transcriber, made possible the preparation of an alphanumeric programming system for the NORC. The system herein described performs the following services: ordering the coding, converting it to machine language, making up and assembling subroutines, and facilitating segmentation. In addition, it checks for a number of errors and allows corrections to be made in a convenient manner.

Coding

The Alphanumeric Compiler is a machine-oriented assembly routine; that is, it is designed primarily to handle instructions written in a form similar to machine language. In addition, it recognizes a group of more powerful instructions called pseudo-codes. These are used for such work as creating and calling subroutines, allocating storage, and setting up segmentation.

Coding to be compiled is written on coding sheets of the form shown on page C3. This is punched on cards, one card per line of coding. The cards are fed into the Universal Data Transcriber, which converts each number, letter, or other symbol into its corresponding CRT Printer digit pair and writes the results on magnetic tape. The tape is used as input to the compiler, which is a NORC program. To reduce ambiguity in handwritten programs the coder should write the letters "I" and "O" as $\mathcal{I}$ and $\emptyset$.

A line of coding consists of 62 alphanumeric characters, arranged in fields from left to right as follows: group tag, 2 characters; symbolic location, 8 characters; command, 4 characters; and 3 operands of 16 characters each.

The compiler orders the coding according to the group tags. On each line of coding, the group tag field should contain either two characters or none. If it contains none, this means that the line is attached to the preceding line for ordering purposes. Any tag may be used several times, resulting in the grouping together of all lines associated with it, in the order in which they appeared in the symbolic input. The two characters may be any combination of letters and numbers but may not include symbols. The compiler orders the tags alphanumerically, i.e., 00--09, 0A--0Z, 10--1Z, ···A0--AZ, ···Z0--ZZ. (Here 0 is a digit, not a letter.) This ordering is easily understandable when the internal representation of the tags is considered, because each two-character tag appears as four digits, and the compiler orders the list of these four-digit tags numerically. In a multi-segment program, the first line of each segment must have a group tag.

Symbolic locations, or alphanumeric names, may be assigned to constants and instructions as needed. It is desirable to assign a name to any instruction which is referenced in another instruction. These names should be meaningful to the programmer; they might, for example, be the titles of corresponding boxes in his flow chart. Names may consist of from one to eight alphanumeric characters. The first character must always be a letter; the others may be any combination of letters and numbers. The names are used only as symbolic locations and not as keys for ordering. No symbolic location may be assigned more than once in a program. The symbolic locations T1, T2,···T19 always refer to the absolute addresses 00021, 00022, ···00039 respectively and must not appear in the location field.

The command field may contain an operation code, a pseudo-code, or the PQ portion of a constant. Many of the operation codes with their most frequently used P fields have alphabetic equivalents as listed on page A1.* Hence the command field for a floating addition instruction may contain 5020, ADD or A, and the resulting absolute code will have 5020 in the PQ field. If 0520 is desired, however, it must be written numerically.

Instructions may be written in three-address form without regard for the peculiarities of the 20,000 word instruction format. The compiler makes two or three instructions out of one where necessary, and it places the correct value in the H field.

Each of the three operand fields may contain a symbolic location, a numeric increment or decrement, and a modifier designation. The numeric part is preceded by + or - and is added algebraically to the absolute equivalent of the symbolic location. The use of a modifier is indicated by a virgule (/) followed by 4, 6, or 8, indicating M_4, M_6, or M_8 respectively. When there is an absolute address with no symbolic portion, the algebraic sign is unnecessary. High-order 0's are unnecessary; i.e., 20 means 00020. The absolute portion and the modifier may be written in either order, because their punctuation distinguishes them.

A hazard must be pointed out with regard to referring to an instruction which has no symbolic location; that is, the fourth symbolic instruction after a line with a symbolic location may turn out to be the eighth line of machine coding past that converted location, because each symbolic instruction may have become two absolute instructions. Therefore, it is best to assign a symbolic name to each instruction which is referred to in another instruction.

As stated previously, the compiler automatically arranges instructions to fit the 20K format and supplies the correct value in the H field when the operation code is in the ranges 20--49 and 60--89. If it is desired to force a value in the H field or to force the PQHJK format when the Q field is outside the above ranges, this may be done by writing Δnn in the R field of the symbolic instruction, where nn is the number to be placed in the H field.

Constants and absolute coding may be written with four digits in each of the four fields (command field and 3 operand fields). Some of the pseudo-codes provide more ways of writing constants.

Pseudo-Codes

The examples of pseudo-codes given in pages B1 and B2 should be used to supplement the following discussion. The examples, of course, are not

*For a complete list of NORC operation codes, see "NORC Programming and Coding Manual."

to be copied exactly but are to be used as a guide. The heavily shaded fields are to be left blank; the lightly shaded fields may be filled in if needed; the remainder must contain information as indicated.

STAR tells the compiler how a program is to be started. The R field contains the number of the segment to be read in first, and the S field contains the location to which control is to be transferred. If a program contains no STAR, the compiler makes up one which starts the program in location 00100 of segment 0001.

FINI is written where a standard stop is desired. There may be several of these in a program.

HALT is used when an error stop is required. This is intended for use with an executive service routine, when a HALT would not be a stop but would be a transfer to an error-printing portion of such a routine.

SEGM is a segment definition statement. There should be one SEGM for each segment of a program, but the SEGM may be omitted from a one-segment program. The segment number, which is written in the R field, may range from 1 to 9998. The S field contains the location over which the first word of the segment is to be read. This location may be an absolute address or a symbolic location from some preceding segment. If the S field is left blank, the compiler prints an error indication and starts the segment at location 00100. If this is where the segment should begin, the error indication may be disregarded. The group tag of the SEGM should be lower than any in the segment, although it may be equal to the group tag of the first line of the segment.

SERD is an instruction to read in the segment whose number is in the R field and transfer control to the location specified in the S field. This location does not have to be in the new segment. This instruction uses the link routine, which is in locations 00008 through 00099 of all compiled programs. The link routine saves the number of the last segment read into memory and reads the program tape forward or backward for a new segment, according to whether the new segment number is larger or smaller. This, of course, assumes that the segment numbers are in ascending order.

WSRD is used to read in a new segment when the segment numbers are not in order or when the program tape has been moved since the last segment was read so that the link routine would search in the wrong direction. WSRD causes the link to rewind the program tape and read forward for the segment.

SAVE is used to reserve a constant number of consecutive locations in memory. The number is written in the R field. The location field must contain a symbolic location. If a SAVE has no group tag, the

compiler reserves a space of the required length immediately after the end of the segment or after the space for the preceding SAVE. If a SAVE has a group tag, however, it is ordered along with the rest of the coding, and the space is reserved at the place where the tagged SAVE appears in the segment.

If a SAVE with a group tag specifies 100 words or less, the program block includes that number of words. If the number is greater than 100, however, the space is not part of the program block as it is written on tape, but it is indicated by a special word at the end of the segment. This word causes the link routine to expand the segment by the required amount at the proper place after the segment has been read in. This is done to avoid writing many unnecessary words on the program tape.

The locations reserved by means of SAVEs almost always contain meaningless information; they should never be expected to contain zeros. As a general rule, SAVEs should be written without group tags. This saves space on the program tape and reduces the time required for segment read-in.

DEFT is used to define names for the "temporary" locations 21 through 39. Definitions may be made in all or any of the R, S, or T fields. A symbolic address followed by "=XX" constitutes a definition, where XX is 21-39. The address thus defined will then refer to the temporary to which it was equated. No absolute part may be used with the address.

The EQUL code causes its symbolic location to be defined and to be assigned the value of the address in the R field. The R field location, if symbolic, must appear in the location field somewhere in the coding.

ROUT is used to call a subroutine and to place the subroutine at the end of the segment if it has not already been placed in the segment. If it is desired to transfer control to some line of the subroutine other than the first, the T field of the ROUT should be written "NAME + n" where NAME is the name of the subroutine and n the desired number of words past the first. If the subroutine is the preset-parameter type and several versions are wanted in one segment, these may be designated "NAME/n" where n is a single digit referring to one version of the subroutine. If the subroutine is an open-ended subroutine, it is inserted in place of the ROUT rather than at the end of the segment.

PUTS instructs the compiler to insert the subroutine named in the R field into the coding where the PUTS appears. This does not set up any call lines to the subroutine.

USUB instructs the compiler not to put the subroutine named in the R field into the segment whose number is in the S field, but to convert ROUTs calling the subroutine in that segment so that they use it from the segment specified in the T field. The user should ascertain that the subroutine is in memory when the S field's segment calls it. In the case of an open-ended subroutine, USUB is meaningless and should not be used.

SUBR is used to create a subroutine from part of a program. All the coding and constants which make up the subroutine should be together in the ordered symbolic program. No absolute addresses should be used in this coding except those that are not to be converted (e.g., print registers). Unchanging constants should be written with the CONS pseudo-code. If the subroutine calls other subroutines, those call lines should be written as ROUTs in the usual way.

The operand fields of a SUBR should contain the following information:

R symbolic - the subroutine name

R absolute (increment portion) -
a. blank if the subroutine is not open-ended and if its call line is to have 60 in the Q field
b. 99 if the subroutine is open-ended
c. the 2-digit Q field to be used in the converted ROUT

S symbolic - blank

S absolute - the block number to be assigned to the subroutine

T symbolic - the letter X followed by the first and last group tags associated with the subroutine's coding

T absolute -
a. blank if the subroutine requires no presetting
b. a number between 1 and 98, showing how many NPUTs are required; 99 if the number of NPUTs can vary

The R symbolic, S absolute, and T symbolic fields must be filled in.

When a SUBR is used, a tape must be on tape code 06. The compiler writes the new subroutine on this tape.

NPUT is a line of input for a preset-parameter subroutine. Its fields should be filled in as specified by the author of the subroutine.

ADDL is used in the presetting portion of preset-parameter subroutines. It states which field of a subroutine line is replaced by which field of an NPUT line.

CONS indicates a constant which is to be placed in a pool of constants for its segment. Each CONS must have a symbolic location. The compiler makes up a list of these constants, eliminates all duplications, and puts the list into the segment after the last subroutine and before the spaces reserved by SAVEs.

CLEV designates a constant which is to be left in the coding where it is written.

The numeric values of CONS and CLEV are handled alike, the difference between them being the way in which they are placed in the program. If a

normalized number is desired, it should be written in the R field with a decimal point. It can consist of as many as 13 digits and may be preceded by a minus sign. A plus sign causes an error printout, but the number is converted properly. Constants which are not to be normalized should be written in the R and S fields of CONS or CLEV, 8 digits per field. A blank field means 8 zeros; a field with less than 8 digits is completed with trailing (low-order) zeros.

CONE and CSIX indicate constants of the type used with the alphanumeric print code, combined option (Q field of 87). The characters written in these codes are converted to their CRTP digit pairs, except for the asterisk, which is interpreted as a space. End-of-word characters (96's) are supplied when needed. CONE should contain no more than 8 characters in the R field and becomes one 16-digit word. CSIX may contain as many as 48 characters, 16 per field, and it becomes 6 words.

CLUE designates a comment by the programmer, and it has no effect on the final coding. CLUEs are retained in the symbolic coding and are listed with it. A CLUE contains up to 48 characters, which are printed exactly as they appear in the coding. Because CLUEs are not edited in any way, the programmer may wish to place hyphens or asterisks between words to give the appearance of spacing.

Error Checking

The compiler checks for a number of errors before doing any compilation. It attempts to correct or remove these errors, and in some cases it replaces a faulty code with a CLUE or a CONE. It prints a description of the error, together with the incorrect and the corrected forms of the instruction. In most cases the compiler can continue its work, so that it can detect several errors in one run. A few types of errors make further compilation impossible, however, so it prints that the program did not compile and stops. Some conditions which the compiler interprets as errors are merely possible errors. If the corrected form is satisfactory to the programmer, he may disregard the error indication.

Printing

The compiler does all its printing on the CRT Printer using camera A, which is normally the Traid camera. The first frame is a film identification, using the deck number taken from the symbolic cards. Following this are the error printouts, if any. A line is printed for each subroutine that is called. Such lines may be intermixed with error lines if mistakes are made in the subroutine area. The last part of the printing is a listing of the entire program. This consists of the ordered symbolic coding on the left side of the page and the resulting absolute program on the right side.

If the compiler is unable to read or write on a tape, it prints "Tape OX bad nn", where X is the tape code and nn refers to the portion of the

compiler which was operating at the time the tape check failure occurred. If the end-of-file indicator is turned on during writing, the compiler prints "Tape OX short nn". After either of these occurrences, the compiler positions all tapes for the next compiling and prepares to stop.

When the compiler must stop without having finished compiling, it prints "Did not compile. Do not run." When it detects errors but is able to complete the program, it prints "Compiled with possible errors." When it finds no errors, it prints "Compiled without errors." One of these three lines should appear at the end of any compilation. The only exceptions to this should be in case of machine malfunction or some tape check failures.

One of the early stages of a compiler run consists of ordering the symbolic coding from tape 01 and writing it on tape 02. When this is finished, the compiler prints "Tape 02 contains ordered symbolic coding." This means that the tape can be used as input to a future compilation, unless "Tape 02 short" or "Tape 02 bad" has been printed.

Tapes

Input to the compiler is on tape code 01, and output is on tape code 02. Tape codes 03 and 04 are "T" tapes, and the compiler is on tape code 09. If corrections are being made, they are placed on tape code 05. If an auxiliary subroutine tape is to be used, it is on tape code 06. The compiler never destroys the information on tape codes 01 and 05.

At the end of a successful compilation, the output tape contains the following:

Standard start block, No. 0000

Ordered (and corrected) symbolic coding

Link block, No. 9999

Absolute coding

Correspondence table, No. 0000

End of file

The standard start block is six words in length and occupies locations 00002 through 00007. The ordered symbolic coding blocks have a maximum length of 8000 words. The link block contains 92 words, occupying locations 00008 through 00099. Each segment of a compiled program

is in one block of absolute coding unless a segment contains 10,000 words or more, in which case it requires two blocks. The block containing the correspondence table is 4,000 words long.

If the program is to be recompiled with corrections, the former output tape becomes the input tape and is placed on tape code 01. If compilation has not been completed but "Tape 02 contains ordered symbolic coding" has been printed, the 02 tape can still be used as input to the compiler, provided "Tape 02 short" or "Tape 02 bad" has not been printed. A new tape is then placed on tape code 02 for the corrected output.

Several programs may be on one tape and can be compiled one after another onto another tape. At the end of each compilation all tapes are positioned properly for continuing.

Corrections

One segment of the compiler makes corrections in the symbolic coding, reorders it if necessary and recompiles the corrected coding. Corrections are written on the alphanumeric compiler coding sheets. A set of corrections may contain no more than 488 symbolic lines. It is placed on tape code 05 and option switch 75 is set to "Transfer".

There are three types of corrections: REPLACE, REMOVE, and NSERT. One of these is written in the R field of a correction line, and the first location to which it applies is written in the S field. The T field specifies the number of consecutive symbolic lines under control of the correction (a blank means 1); the correction, if a REPLACE or NSERT, is followed by this number of lines of coding.

As the names imply, REPLACE calls for a line-by-line replacement of old coding by new, starting with the line specified in the S field of the REPLACE; REMOVE calls for T consecutive lines to be removed, starting with the location specified; NSERT calls for T lines to be placed in the symbolic coding between the location specified and the one following it.

It must be noted that the corrections refer to symbolic lines, regardless of their absolute equivalents; hence the correction "REMOVE B+6" causes the sixth symbolic line after location B to be removed.

The absolute part of the S field of a correction specification must not contain a minus sign or a value greater than 0999. If the first line of coding has no symbolic location, it may be specified in a correction line by a 1 in the S field. The next line may be specified by 2, etc. If it is desired to insert coding before the first line, the NSERT instruction should have a blank S field. When the compiler finds a correction line with no symbolic location in the S field, it prints an indication of a possible error and uses the line as described above.

If one symbolic location has been assigned to two lines of coding and it is desired to correct this error by changing the symbolic location of the second line, this line must be specified in the REPLACE instruction

as the preceding symbolic location plus the number necessary to reach the faulty line.

If a program has been compiled previously and the ordered symbolic tape has been written, this tape should be used for input to a correcting and recompiling run. A listing of the ordered symbolic tape should be used in planning corrections, because the original coding is probably unordered and its use at this time could cause errors.

Restrictions

There are limitations on the number of times some pseudo-codes may occur in a program. These are as follows:

Code	Maximum
STAR	1
SEGM	50
SERD, WSRD	Total of 100
SAVE	100 per segment (SAVEs with group tags and with R field values less than or equal to 100 do not count toward this maximum.)
DEFT	50
USUB	50
SUBR	5
NPUT	62 per segment (More may be used in connection with open-ended subroutines or with PUTS codes.)

No more than 2000 symbolic locations, including those in subroutines, may be used in a program.

A maximum of 1300 group tags may be used.

The pseudo-code NPUT must follow ROUT, PUTS, or another NPUT in the ordered coding.

The following pseudo-codes must have symbolic locations: SAVE, EQUL, CONS.

Subroutine System

The Library

The subroutine library appears on the compiler program tape (t.c. 09) immediately following the compiler program and its end of file.

The first block (block 0000) contains information about each subroutine, such as the block number corresponding to the subroutine name and the list of subroutines called by it. The subroutines, in a modified symbolic form, follow this block in order of their block numbers.

The programmer may also use an auxiliary library tape on tape code 06. This tape is of the same form as the main library, and will contain subroutines created by SUBRs in the present compiling as well as some of those created by previous compilings. If there are any SUBRs in a program, there must be a tape on tape code 06. If there are subroutines already in the auxiliary library that are to be preserved, option switch 74 must be set to "Transfer".

The Use of Subroutines

An ordinary program-parameter subroutine is called by a ROUT. The ROUT is translated by the compiler into the appropriate call line to the subroutine. It also causes the subroutine to be entered into the coding of the segment. If there is no ROUT for the subroutine in a segment, it still appears there if it is referred to by a PUTS or a USUB, or if it is called by another subroutine in that segment. If it is referred to by a PUTS, it appears in the coding in place of the PUTS. Otherwise, it appears at the end of the segment in order of its block number.

The above statements also apply to preset-parameter subroutines with the exception that a USUB does not force the subroutine to be present. The proper set of NPUT lines must appear with a ROUT or a PUTS to the subroutine somewhere in the segment in which it is to appear. It need not be with the first ROUT to this subroutine but must be before or with the first PUTS of the subroutine in the segment. The preset-parameter subroutine appears at the end of the segment or in place of the PUTS.

The use of a version number allows a subroutine to be included in a segment as many as ten times. The compiler treats these different versions as completely independent subroutines. Thus the same preset-parameter subroutine can be included in a segment several times with independent call lines and parameterization.

Open-ended subroutines are fragments of coding which are essentially stripped down subroutines. An example might be a square root subroutine imbedded in the main program starting at location M which receives the argument in register storage as a result of operating line M-1, computes the square root, leaves it in register storage, and exits to the line immediately following its last line. An open-ended subroutine may or may not be a preset-parameter subroutine. (In the example above, it might be profitable to have the result stored in a preset location.)

An open-ended subroutine is caused to be inserted by either a ROUT or a PUTS referring to the subroutine. In either case the call is replaced by the subroutine. If the subroutine is also a preset-parameter

subroutine, the ROUT or PUTS must be accompanied by the appropriate NPUT(s). Such a subroutine cannot be referred to in a USUB. An open-ended subroutine may be inserted in a segment as many times as desired without distinguishing the calls by version numbers. In fact, such version numbers are ignored.

Open-ended subroutines are created by SUBRs in the conventional manner. The SUBR for such a subroutine is distinguished by a +99 in the absolute R field.

Creating New Subroutines

New subroutines are added to a programmer's auxiliary library tape by means of SUBRs. A SUBR causes revision of the information block at the beginning of the auxiliary library, removes any subroutine whose name or block number is the same as the newly created subroutine, and places the new subroutine on the auxiliary tape in order of its block number. The coding from which the new subroutine was created remains in the program, but any call to this subroutine will cause it to be inserted according to the conventional rules as well.

If option switch 74 is on "Off", the auxiliary library is destroyed and a new one developed consisting of only those subroutines defined by SUBRs in the coding being compiled.

There are very few limitations on the writing of subroutines. Any of the alphanumeric and numeric operation codes may be used as can most of the pseudo-codes. The following pseudo-codes may be used as they would be in ordinary coding: FINI, HALT, SAVE, NPUT, CONS, CLEV, CONE, CSIX, CLUE.

STAR, USUB, and SUBR may appear in the region from which the subroutine is to be made up, but they are omitted from the library tape coding and apply only to the main program. An EQUL may be used but must refer to a location in the subroutine or to an absolute location. A DEFT applies to both the subroutine and the main program. However, all locations referred to in this manner are converted to the appropriate absolute locations in the range 21-39 and the DEFTs removed before the subroutine is written on the auxiliary tape. SEGM, SERD, and WSRD are illegal, are omitted, and cause an error printout. ROUT and PUTS may be used with their usual meaning but are subject to restrictions. Any version number is ignored. Preset-parameter subroutines which are not open-ended may not be called. Open-ended and program-parameter subroutines may be called by a ROUT. Only open-ended subroutines may be called by a PUTS. If the above restrictions are not followed, the offending ROUT, PUTS or NPUT is omitted and there is an error printout.

Ordinary program-parameter subroutines called in another subroutine appear in the segment with the calling subroutine. If there is a PUTS in the calling subroutine, the called subroutine appears in the coding in place of the PUTS.

A subroutine being created may call another being created in the same coding if the called subroutine is not open-ended and if it has no dependencies itself.

New subroutines on an auxiliary tape may be added to the library on the compiler program tape only by means of a special program.

Preset-parameter Subroutines

Preset-parameter subroutines are like other subroutines with the exception that some modifications to lines of coding in the subroutine are effected in the process of inserting the subroutine into the program being compiled. These modifications are specified by means of the ADDL pseudo-codes in the subroutine in conjunction with the NPUT lines associated with a call to the subroutine.

These ADDL pseudo-codes specify where in the coding of the subroutine these changes (replacements) are to be made and where in the NPUT lines the changes come from. Thus, an ADDL might in effect say, "replace the symbolic S field of line PØT + 7 with the symbolic T field of the second NPUT line."

In creating a preset-parameter subroutine, the programmer attaches the required ADDL lines to the end of the subroutine. These lines do not appear in the compiled program but do appear as a part of the subroutine on the auxiliary tape. When the subroutine is called by a program, the ADDL codes likewise do not appear in the coding.

The form of the ADDL code is as follows:

Field		Contents
R-Symbolic	-	NPUT
R-Absolute	-	blank if 1st NPUT is being referred to. +n if the n + 1 st is being referred to.
S-Symbolic and Absolute	-	line of coding in the subroutine being referred to.
T-Symbolic	=	XXYY

XX = RS, RA, SS, SA, TS, TA
YY = PQ, RS, RA, SS etc.

XX specifies the appropriate field of the specified NPUT line.

YY specifies the appropriate field of the specified subroutine line.

If YY = PQ the corresponding NPUT entry must be of the following form:

1. if it is an absolute field
   XXXX or ± XXXX

2. if it is a symbolic field
GGXXXX

where XXXX is the desired numeric operation code and GG is an arbitrary pair of alphabetic characters

The above case allows the operation code of a line of a subroutine to be a preset parameter.

The following example may help clarify the creation and use of preset-parameter subroutines. The program on page 14 creates an open-ended preset-parameter subroutine named "FIXIT" and calls it. In the process of compiling, the subroutine is added to the auxiliary tape. The program will behave as if it had been coded as shown on page 15.

APPENDIX A

ALPHABETIC EQUIVALENTS OF NUMERIC OPERATION CODES

5020	ADD or A
5021	ADDN
5022	SUB or S
5023	SUBN
5024	MUL or M
5025	MULN
5026	DIV or D
5027	DIVN
5028	ABS
5030	SADD
5031	SADN
5032	SSUB
5033	SSUN
5034	SMUL
5035	SMUN
5036	SDIV
5037	SDIN
5038	SABS
5039	TRTR
0040	MADD or L
0041	MSUB
0042	EXTR or X
0049	MSSI
0060	TRAN or T
0061	STOP
0062	ROND
0063	BRAN or B
0064	TROV
0065	TRAD
0066	TRZR
0067	EOF
0068	TCF
0069	TRPR
0070	JIFO or E
0071	SIFO
0072	JIFN or U
0073	SIFN
0080	PRLO
0081	PRTA
0082	PRTB
0083	SFPA
0084	SFPB
1085	PNUT
2085	PNUK
1086	PALT
2086	PALK
1087	PANT
2087	PANK
1188	POST
2188	POSK
1688	PLOT
2688	PLOK
XX98	WIND (XX)

APPENDIX B

PSEUDO-CODES FOR THE ALPHANUMERIC COMPILER

Note: Unshaded fields must be filled in.

FORMAT						MEANING TO COMPILER	RESULT IN COMPILED PROGRAM
Group Tag	Location	Command	R Operand	S Operand	T Operand		
		STAR	20	FIRST		Set up the link block so that program will be started in location FIRST of segment 20.	Word in link block.
		FINI				Make up a standard stop.	L: 0080 60 (L+1) 00001 L+1: 0061 00 00000 00001
		HALT				Make up an error stop.	0061 40 00000 00000
CA		SEGM	10	COM-2		Begin segment 10 at group CA and make its first absolute address the same as that of COM-2.	New block of coding.
		SERD	30	HDOT		Set up lines which will transfer to the link and will cause it to read segment 30 and transfer to HDOT.	L: 0060 39 (L+1) 00010 L+1: NNNN OX MMMMM SSSSS (N = seg. no.; X = 8 if seg. length > 10,000; M = first loc. of seg.; S = starting loc.)
		WSRD	30	HDOT		Same as above but transfer is to different place in link so that program tape is re-wound before segment is read.	L: 0060 39 (L+1) 00008 L+1: NNNN OX MMMMM SSSSS
	GAMMA	SAVE	300			Reserve 300 consecutive locations, the first one to be named GAMMA. If there is no group tag, reserve this space at the end of the segment.	
		DEFT	C1 = 31	TS4=22	K8 = 31	Define symbolic names for locations in the group 21-39.	
	ALPHA	EQUL	BETA+5			Assign the absolute location for BETA + 5 to ALPHA.	
		ROUT			ARCTAN	Set up a call line to the ARCTAN subroutine and place the subroutine at the end of the current segment if it is not already there.	L: 05XX 20 (L) (R) X is specified in sub-routine; R = first line of subroutine.

(Continued on next page)

FORMAT						MEANING TO COMPILER	RESULT IN COMPILED PROGRAM
Group Tag	Location	Command	R Operand	S Operand	T Operand		
		PUTS			SINCOS	Put the SINCOS subroutine into the coding at this point.	
		USUB	SQRT/2	15	25	In segment 15, convert any call lines to the second version of the SQRT subroutine to use that subroutine from segment 25.	
		SUBR	SIMPSON + 65	3	XGAHB+4	Make up a subroutine from the coding associated with group tags GA through HB; give it the name "SIMPSON" and block number 0003; make an entry for it in the subroutine table on the auxiliary tape, noting that 65 goes in the Q field when converting ROUT's which call this routine; write this subroutine on the auxiliary tape; it is a preset-parameter routine requiring 4 NPUT's.	
		NPUT				This is input to a preset-parameter subroutine. (The R, S, and T fields are filled in as specified by the author of the subroutine.)	
		ADDL	NPUT + 2	ARC + 5	SSRS	Set up the presetting portion of the subroutine being made up so that the symbolic part of the S field of the third NPUT will be placed in the symbolic R field of line ARC + 5 in the subroutine.	
	KEY	CONS	12345	77		Place this constant in the constant pool for this segment if it is not already there.*	1234500077000000
		CLEV	-356.2000198			Place this constant in the program where it appears.*	0213562000198000
		CONE	ABCD123			Make this a one-word constant suitable for use with an 87 code (combined alpha-beta print).	5152535441424396. The 96 is supplied when needed.

*If it contains a decimal point make a normalized constant, using the first 13 digits of the R field. Otherwise take 8 digits from R and 8 from S, supplying trailing zeroes where there are less than 8 digits.

(Continued on next page)