

UNCLASSIFIED

NRL Report 5974  
NAREC Reference #30

# NABUR

## A NAREC Assembler for the Burroughs D825 Modular Data-Processing Computer

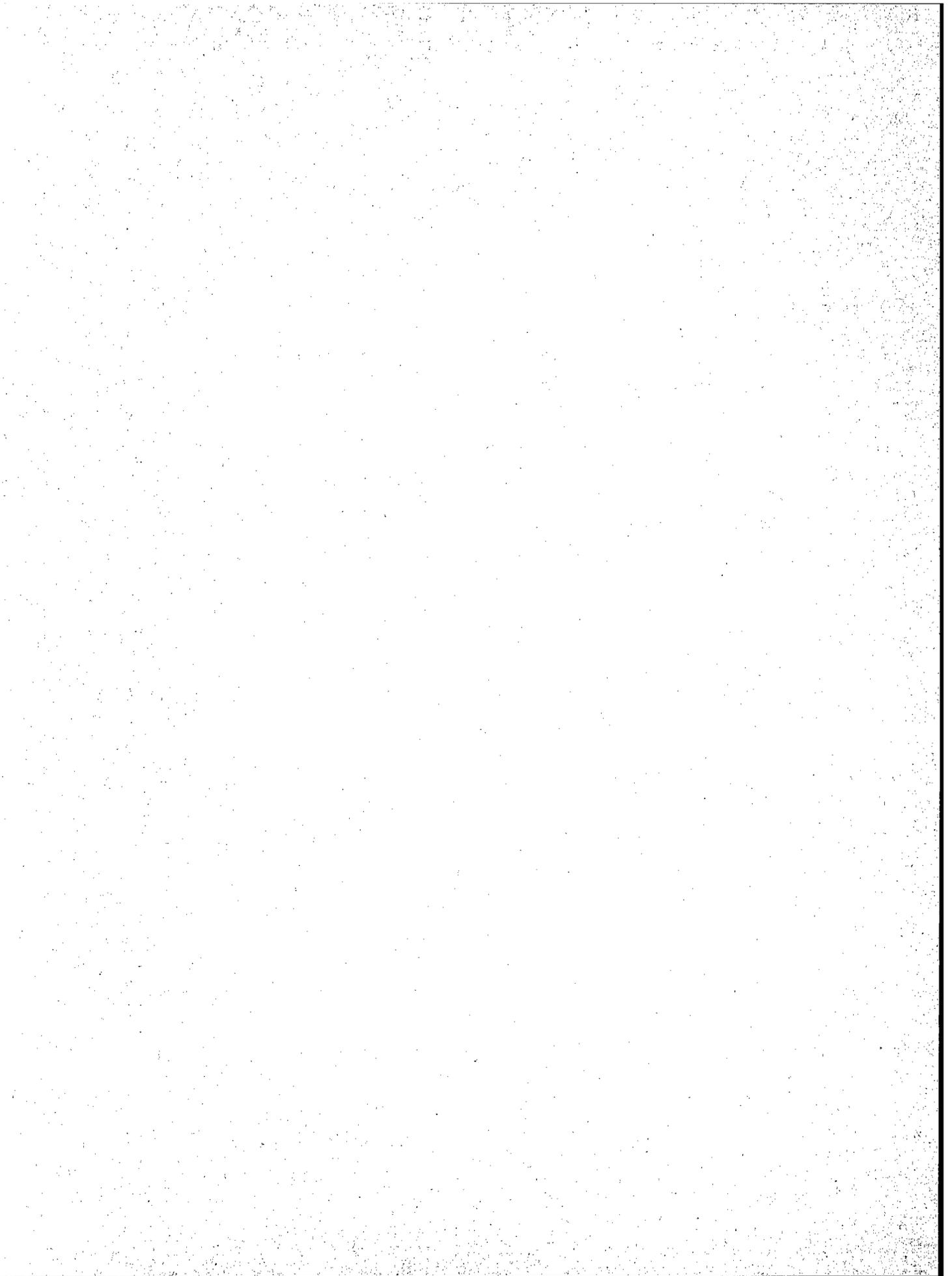
ROBERT M. MASON AND IRENE G. FISHMAN

*Applied Mathematics Staff  
Office of Director of Research*

January 13, 1964



U.S. NAVAL RESEARCH LABORATORY  
Washington, D.C.



## CONTENTS

Abstract .....	1
Problem Status.....	1
Authorization .....	1
 INTRODUCTION.....	 1
 PART I – NABUR LANGUAGE .....	 1
1. Preliminary Discussion .....	1
2. Programming Sheets .....	3
3. Symbolic Addresses or Labels .....	8
4. Regular or Quasi-Regular D825 Instructions .....	10
5. Triadic Forms .....	10
6. Operator Syllables.....	11
7. Index Syllables .....	12
8. Basic Address Syllables (Memory and Branch).....	12
9. Special-Use Syllables.....	13
10. Pseudo-Instructions.....	15
11. NABUR Operator Instruction Sheet .....	18
12. Source Tapes .....	18
13. Error Statements.....	18
 PART II – THE NABUR ROUTINE.....	 19
1. Name of Program .....	19
2. Class .....	19
3. Purpose .....	19
4. Language Used.....	19
5. Authors.....	19
6. Formal Statement.....	19
7. Operator Instructions .....	24
8. Output .....	24
9. Tape Labels .....	25
10. Address Information .....	25
11. External Working Memory.....	25
12. Versatility .....	25
13. Remarks .....	25
 ACKNOWLEDGMENTS .....	 26
 REFERENCES.....	 26
 APPENDIX A – Numerical Listing of Burroughs 825 Instructions.....	 40
 APPENDIX B – Block Diagrams of Key Subroutines .....	 41
 APPENDIX C – Summary of NABUR Rules .....	 48

# NABUR

## A NAREC Assembler for the Burroughs D825 Modular Data-Processing Computer

ROBERT M. MASON AND IRENE G. FISHMAN

*Applied Mathematics Staff  
Office of Director of Research*

NABUR, an assembly routine written for the Naval Research Electronic Computer (NAREC), produces object programs suited to the Burroughs D825 Modular Data-Processing Computer. The input data for this routine may consist of one or more punched tapes coded in a source language that augments D825 basic machine language. This report discusses conventions to be followed in writing source programs and general aspects of the routine itself. A numerical listing of Burroughs D825 instructions, block diagrams of key subroutines, and a summary of NABUR rules appear as appendices.

### INTRODUCTION

This report concerns an assembly routine called NABUR, written for the NAREC (Naval Research Electronic Computer), that permits the use of this machine in the automatic assembly of object programs for running on another of NRL's computers, namely, a Burroughs D825. The D825 Modular Data-Processing System has been developed to perform command-and-control functions in a large military man-machine complex. The initial system, constructed for NRL with the designation AN/GYK-3(V), has now been installed and tested. Reference 1 is a paper reviewing the design-criteria analysis and design rationale that led to the system structure of the D825.

Part I of this report states the rules of NABUR language. These rules must be followed explicitly in writing D825 source programs for NAREC assembly. Part II explains the assembly program's basic features, so that later improvements can be made to the NABUR system, if required. It is of interest to note that the NABUR program has also been written in an assembly language, namely NAR (NAREC Assembly Routine) symbolism.

Although the NABUR program bears some internal resemblance to the Burroughs 220/D825 Modular Processor Assembler (*e.g.*, at one stage in its development it copied its error statements from this assembly program verbatim), it differs markedly in regard to basic conventions. The symbolism employed in writing NABUR source

programs has been planned to conform closely to the symbolism of the Burroughs D825 machine language. Thus, all combinations given in Ref. 2 are retained to designate D825 operations (these are listed in Appendix A), and only combinations consisting of three letters are introduced to represent quasi-regular and pseudo-instructions. The use of these terms will be made apparent in later sections of this report.

The underscore symbol, *i.e.* "\_\_\_" applied to a numeral is used in this report to indicate decimal numbers, unless they appear as exponents or subscripts or are otherwise unambiguous. No such indication is used in describing outputs, either in listing NABUR error statements or in illustrating the checking-output format. The comma symbol, *i.e.* ",", after a numeral is used in this report to indicate octal numbers only where it is necessary to show the correct usage of the comma according to the rules of NABUR source language.

## Part I NABUR LANGUAGE

### 1. PRELIMINARY DISCUSSION

A "NABUR source program" is a routine handwritten in symbolic (*i.e.*, NABUR) form. Such a program is typed on one of several NAREC Flexowriters using so-called standard six-bit NAREC character representation to obtain a punched tape called a "NABUR source tape." One or more of these source tapes are read into the NAREC memory at assembly time in order to arrive at

NRL Problem R06-07; Projects RF.001-08-41-4552 and SF 001-08-01, Task 9249. This is an interim report on one phase of the problem; work on this problem is continuing. Manuscript submitted May 23, 1963.

the subject data which is handled internally by the NABUR assembly routine. The end product of the assembly process is a D825 machine-language program called the "object program." The output consists of a bioctal punched paper object tape for input to the Burroughs D825 computer and a hard copy of the checking output.

The NABUR assembly process therefore splits up into two important phases: input and output. The input phase precedes the first of two passes made by the assembler through the subject data. It involves reading in first the NABUR system tapes, and second, the source tapes. That part of the output phase that produces an object tape for later input to the D825 will follow an assembly if the assembly is successful. Otherwise, an object tape is produced only by special request. The object phase also includes some line printing that occurs (a) during readin, (b) at midpass, and (c) at regular intervals during the second pass. It also may include a printout of error statements during either the first or second pass, or both. It is not necessary to punch out a tape during this preliminary part of the output phase, but it is possible to do so if this is requested as a special instruction on the NABUR Operator Instruction Sheet.

At readin, printout consists of a listing of heading and check sum for each source tape. This information will be checked for accuracy by the NAREC machine operators, if they are advised of the expected values of the check sums in advance. In case of a discrepancy, the operators then can take necessary steps to ensure a correct run, either by rereading the tape in question to try to obtain the correct sum, or if this fails, by returning the problem folder to the programmer for further study. At midpass, printout consists of a listing of labels and their octal equivalents. Some of these labels were reserved in advance by the NABUR system to relate actual addresses in thin film to the common names of certain D825 registers or to render binary meanings to certain mnemonics helpful in stating transmit modifier syllables or 16-bit absolute addresses. Others have been reserved in name only (semireserved) so that the programmer may tell the assembler what he is placing in the BAR, BPR, or SAR registers. The remainder are labels that have been either preassigned, defined, or assigned to NABUR words as symbolic addresses by the programmer. During the second pass, printout

consists of a checking output in which D825 octal coding worked out by the assembler appears alongside a restatement of the original lines of the source program. A more complete discussion of the nature of NABUR output is to be found in Part II, Section 8 of this report. At the successful conclusion of an assembly, printout consists of a most inscrutable six-bit character representation of the bioctal object tape which serves primarily to assure the programmer that his assembly was successful and that an output tape was punched.

On its first pass through the subject data the assembler must search and partially decode the information available to it in order to associate a numerical value with each symbolic address or label. At this time the assembler begins to form a pre-image of the D825 object program in the NAREC memory. It does this by creating a skeletal frame on which to hang the object data. As each operator syllable is formed it is placed in this frame. As each nonoperator syllable is discovered, a blank syllable consisting of twelve binary-zero digits is inserted in the frame, but only for the time being. During the first pass the assembler may find a few obvious mistakes, such as the same symbolic address assigned more than once or an erroneous character punched on the tape. If so, it will record one or more suitable error messages, for example, the statement "duplicate label," or the statement "improper number of spaces in the specification field." If any critical mistake is found during the first pass, the assembly process will stop at midpass, after first listing the labels used and indicating the occurrence of a "bad stop" by printing five asterisks, *i.e.* "\*\*\*\*\*," on the output sheet. Otherwise, the assembly process will continue.

On its second pass through the subject data the assembler tries to complete the object program started earlier, by filling in the missing pieces of object data in their proper places. This object data is injected (one, two, three, or four consecutive syllables at a time) by means of a "Store Q" subroutine. A block diagram of this subroutine appears in Appendix B. Further checking for illegalities is carried out during the second pass, notably for exceeding the maximum size limits on components for syllables, writing octal or decimal numbers improperly, etc.

Sometimes several diagnostic messages will bracket a single mistake and help the programmer

to pinpoint it. However, there is no guarantee that every mistake contained in the source material will be turned up by the assembly routine, for logical errors often masquerade as legitimate source statements. During the second pass the object program is completely filled in, if it is possible to do so. Provided no mistakes are found, the assembly process concludes by punching out a copy of the object program on paper tape. This tape contains the machine-language D825 object program that has been produced. This program is punched in the same character coding as that provided by the Flexowriter model associated with the D825 computer.

## 2. PROGRAMMING SHEETS

Figure 1 shows a sample NABUR Programming Sheet. At its top, space is provided for stating the title of the problem, the programmer's name, the date of completion of the sheet, the RCC problem number, the tape label assigned, and finally the page number. The remainder of the sheet is divided into rows, corresponding to NABUR source lines, and columns, corresponding to various chunks of information which go to make up complete source statements.

Tape feeds and code deletes are ignored by the NABUR source-program input routine;\* hence, they are not counted as characters. All other characters in the source programs, including format characters (carriage return, tab, and space), will enter the NAREC memory. Recognition of format characters is basic to the assembly process. Because format characters play a fundamental role in keeping the assembly routine on the right track, rules for their use must be carefully observed.

Excluding comment lines, which do not reappear in the checking output, each line of NABUR source programming contains one and only one instruction. Sometimes this instruction will be regular or quasi-regular, leading to insertion in the object program of an operator syllable followed by at most six other syllables. At other times, it will be a pseudo-instruction, perhaps FLO (floating point) or INT (integer), not calling for insertion of an operator syllable, but rather for insertion of four data syllables; or, in the case of SYL (syl-

lable), calling for insertion of a single data syllable. Less often, it will be a pseudo-instruction notifying the assembler of something of which it must take cognizance, but not leading to insertion of any syllables in the object program at all.

At the beginning of any line, the character "carriage return" followed by an "m" will indicate the end of that particular source program (there may be more to follow); the character "carriage return" followed by the character "tab" will indicate that the remainder of the line represents a comment\* or the continuation of a comment; and the character "carriage return" followed by any character other than those mentioned above will indicate the start of a line designation. Additional carriage returns will always have the effect of re-initializing for the beginning of a line.

Three vertical double bars cross the NABUR programming sheet (Fig. 1), dividing each line into compartments. These double bars represent "tabs." The first two of these tabs enclose and set off the symbolic address assignment field, which may be left vacant; the remaining tab terminates the address specification field. Four vertical single bars also appear on the NABUR programming sheet. These single bars represent "spaces." The first space follows the operation field and must be there, even though the operation in question requires no address groupings (e.g., SSU—step stack up). In this case the remaining spaces are omitted by the typist, and the tab signified by the last double bar comes immediately after the space following the operation symbol. The next two spaces act to separate address-grouping fields. The second one is omitted if address grouping III is not entered, and both spaces are omitted if and only if address groupings II and III are both not entered. The tab signified by the last double bar immediately follows the last address grouping entered, if any. A space is typed (along with the remark that follows) only if the letter "r" appears in the remark indication field labelled "r." Remarks reappear in the checking output.

These format requirements are summarized below for instructions having three, two, one, or zero address groupings.

(3) CR Line TAB Symb. Addr. TAB Op. SPACE  
**A1** SPACE **A2** SPACE **A3** TAB r SPACE  
 Remarks CR

\*A. B. Bligh and I. G. Fishman, "Alphanumeric Tape Readin Subroutine," NAREC Bulletin No. 60, April 22, 1963.

\*A distinction is made among "comments," "remarks," and "asides" by the NABUR system. The exact differences in the meaning assigned to these terms are stated explicitly on Page 8.



(2) CR Line TAB Symb. Addr. TAB Op. SPACE  
A1 SPACE A2 TAB r SPACE Remarks CR

(1) CR Line TAB Symb. Addr. TAB Op. SPACE  
A1 TAB r SPACE Remarks CR

(0) CR Line TAB Symb. Addr. TAB Op. SPACE  
TAB r SPACE Remarks CR

Symbolic addresses and remarks often are omitted.

The left-most column of the NABUR programming sheet is used for line designations. Usually these are decimal line numbers, but as long as they include one non-tab character and do not start with an "m," "tab," or "carriage return," their composition is arbitrary. In normal practice, a source line will consist of a line designation followed by a tab followed optionally by a symbolic address followed by a tab; then will come a NABUR instruction word. This word will always begin with a three-letter operation symbol followed by one space and as many as three address groupings, each separated by one space. This word will always end with a tab. Theoretically, there is no limit to the number of characters in a NABUR word. After a NABUR instruction word possibly comes an "r," a "space," a remark, and then a "carriage return." If there is no remark, the last tab is immediately followed by the carriage return.

The columns headed "Address Grouping I," "Address Grouping II," etc., correspond exactly to the syllable layout of D825 instructions given in Appendix D of Ref. 2. When the operation at hand has nonoperator syllables associated with it, those columns corresponding to the positions of these syllables in the layout are to be filled in by the programmer. All other columns are to be left empty. Prior to the first address grouping and immediately after the operation symbol, a space will be inserted by the typist. A space will also be inserted between each adjacent pair of address groupings. The last address grouping in a line will be immediately followed by a tab. No space or tab characters are permitted within an address grouping.

All numbers appearing in address groupings are considered to be decimal integers, if not immediately followed by a comma character. Otherwise, they are to be taken as octal integers. NABUR takes care of conversions to binary. Octal integers are translated, character by character, from 6-bit to 3-bit form to obtain binary; decimal integers are translated, character by character, from 6-bit to 4-bit form and then converted to ob-

tain binary. Leading binary zeros are added as necessary.

Figure 2 is a reproduction of a portion of a source program as it originally appears on the NABUR Programming Sheet. Figure 3 shows the complete typed copy of the corresponding source tape which was obtained as a byproduct during tape preparation. For comparison, the output example given in Part II, Section 8, includes this same source program as one of four that were assembled together using the NABUR program. An explanation of this output must be postponed until that section is reached.

The following terminology is used throughout the remainder of this report in dealing with the subject of address groupings.

### ADDRESS PREFIX

Either a leading "+" (sometimes understood) or a leading "-" is carried by an address grouping to indicate how the following quantities are to be interpreted. This interpretation may be summarized as follows:

<u>Address Grouping Pertains to</u>	<u>Meaning</u>
M - Memory Syllable	The following quantity is a direct address (+) or an indirect address (-).
S - Shift Syllable	The following quantity is a count of leftward (+) or rightward (-) binary displacements.
Ia - Index Increment Amount Syllable	The following quantity is an increment (+) or decrement (-).

A leading "slash" (or "oblique") character, *i.e.*, "/", is used to introduce a triadic form. (See Part I, Section 5.)

Name		Title		Date	Prob. No.	Tape Label	Page
<i>Campbell</i>		<i>arcinp/arcoss subroutine</i>		<i>1-31-63</i>		<i>4815</i>	<i>1</i>
LINE	SYMB. ADDR.	OP.	ADDRESS GROUPING I	ADDRESS GROUPING II	ADDRESS GROUPING III	REMARKS	
0	arc1	stf	lpr	n			
1		lff	n	bar			
00		df	intpr	arc1			
01		df	inbar	arc1			
2		stw	sr	n			
3		ltw	n	+74,			
4		load	r	n	n		
5		lff	n	17,			
6		esl	n	+1	n		
7		trb	n	n	arc1 2		
8	arc1	trm	n	pu	n		
9		cls	r	arc1	arc1 3		
10		cgr	r	arc2	arc1 4		
11		lmu	r	n	n		
12		lsw	n	arc3	n		
13		dsc	n	+2	n		
14		lmu	r	r	n		
15		lsw	arc1 5	r	n		
16		lsw	arc1 4	n	n		
17		lad	n	arc1 3	n		

Fig. 2 - Excerpt of handwritten source program. Note correspondence with first 17 lines of Fig. 3.

;;;;

4815 arcsin/arccos subroutine campbell/ej

nnnn

```

0      arcb      stf bpr n
1      ltf n bar
00     def inbpr arcb
01     def inbar arcb
2      stw srr n
3      ltw n +74,
4      bad h n n
5      ltf n 17,
6      es1 n +1 n
7      brb n n arcb2
8      trm n pu n
9      arcb1     cls h arcc1 arcb3
10     cgr h arcc2 arcb4
11     bmu h n n
12     bsu n arcc3 n
13     dsa n +2 n
14     bmu h h n
15     bsu arck5 h n
16     bdv arck4 n n
17     bad n arck3 n
18     bsu n n n
19     bdv arck2 n n
20     bad n arck1 n
21     bmu n n n
22     bad n arcc4 n
23     uct arcb6
24     arcb2     xuc +1 //17,/0, arcb1
25     arcb3     bmu h h n
26     bsu arck5 h n
27     bdv arck4 n n
28     bad n arck3 n
29     bsu n n n
30     bdv arck2 n n
31     bad n arck1 n
32     bmu n n n
33     uct arcb6
34     arcb4     stf 17, n
35     ltf n 37,
36     ltw h +160,
37     bsu arcc5 n n
38     srj 3, 0,
39     arcr1     stf 37, n
40     ltf n 17,
41     stw +160, n
42     bmu n arckb n
43     bsu arcka n n
44     bmu n n n
45     bsu arcc6 n n
46     stw +74, n
47     ltw n srr
48     arcb6     uct arcb7/17,
49     arcb7     srr
50     ab71      trm n xu n
51     srr
52     ab72      bsu arcc6 n n
53     srr
54     ab73      bad arcc6 n n
55     srr
56     arcc1     int 132404 746316 3255,
57     arcc2     int 166203 674746 1000,
58     arcc3     int 040000 000000 0000,
59     arck5     int 132465 255632 1603,
60     arck4     int -005241 026062 4515,
61     arck3     int 343532 000065 6622,
62     arck2     int 031201 621751 2156,
63     arck1     int 041463 102013 1753,
64     arck2     int 062403 443722 4334,
65     arck1     int 103146 204026 3726,
66     arcc4     int 062207 734244 3343,
67     arcc5     int 200000 000000 0000,
68     arcka     int 212716 304511 6426,
69     arckb     int 025641 515104 4031,
70     arcc6     int 144417 665212 2212,

```

mmmmmm

Fig. 3 - Typewritten source program

### ADDRESS UNIT

An unsigned, uncompounded element may be either (a) a decimal or an octal integer, or (b) a label written in accordance with the definition given in Part I, Section 3. If it is an integer, call it  $x$ , then it must satisfy the inequality  $0 \leq x \leq 4095$ , unless otherwise stated. If not, an error will be noted.

#### EXAMPLES

3891  
sam  
17,

### ADDRESS QUANTITY

An unsigned, possibly compounded element consists of one or more address units joined by "+" or "-" concatenation symbols. These symbols operate as "plus" or "minus" signs, respectively.

#### EXAMPLES

3891+17,-sam  
17,+3891-sam  
0-sam+3891+17,

NOTE: All of these expressions will produce the same numerical value.

### ADDRESS GROUPING

Either a "+" prefixed address quantity is followed by three or less "/" prefixed address quantities, or a "/" prefixed address quantity is followed by five or less "/" prefixed address quantities. An address grouping is terminated by the first space or tab character; an address quantity, by the first space, tab, or slash character; and an address unit, by the first space, tab, slash, plus, or minus character.

### ADDRESS ARITHMETIC

This is a feature of NABUR assembly that is provided by a fundamental subroutine, called "Adar." See the block diagram of this subroutine given in Appendix B. The expression

"-sam+pete-3,..."

appearing at the beginning of a memory syllable address quantity has the following interpretation. (a) The leading "-" sign indicates that the following quantity is to become an indirect address. (b) Both "sam" and "pete" are unreserved labels having numerical values determined by the assembler during the first pass, and for the rest of the assembly these values are to be found in the table of octal correspondences. (c) Both "pete" and "3" are "address modifiers," whereas "+pete" and "-3" are successive "modifications" of sam. (d) The plus sign "+" and minus sign "-", are concatenation symbols used to form the compound address quantity. The Adar subroutine places no restriction on the number of modifiers that can be used in writing a single address quantity.

The last column, headed "Remarks," is provided for the convenience of the programmer. The typist will not punch on the source program tape what is written in this column, unless the letter "r" (which is typed too) appears in the remark indicator field. Thus, the last column will contain (a) nothing, (b) an aside, which is not typed, or (c) a remark, which is typed and reappears in the checking output. To have the source tape contain a passage of explanatory text that is felt to be too long to be a part of the checking output, simply initiate the line with a "CR tab" combination. Such a line will constitute a "comment." Comments differ from asides in that they are typed on the source tape; they differ from remarks in that they do not reappear in the checking output.

### 3. SYMBOLIC ADDRESSES OR LABELS

Some operator syllables must occupy the left-most syllable position of a computer word. For example, repeated instructions must be stored with their operator syllables occupying a leading position. In order to accomplish this through assembly, each such instruction must be marked in the source program. The mark is a "symbolic address."

A NABUR word is assigned a symbolic address by making an appropriate entry on the NABUR programming sheet in the column headed "Symb. Addr." and adjacent to the word itself. Each such entry must meet the following definition of a label. A "label" is a string of five or less strictly alphanumeric characters containing at least one unmistakable letter somewhere in the string (*i.e.*,

some alphabetic character other than the letter "l"). The programmer should also be careful to avoid confusion between the letter "o" and the numeric character "0," which have different coding. A certain number of such strings, some called "reserved labels" and others called "semire-served labels," are already present in the NABUR program table of octal correspondences and therefore cannot be used for symbolic addresses by the programmer. These are listed in Table 1. Although permissible, it is not advisable for symbolic addresses to be chosen so that they spell out operator symbols.

EXAMPLES

valid: b2b5 d alpha  
 w1 111t1  
 not valid: snag bar stk4 xr  
 not advisable: fad cla nop

If a source line is assigned a symbolic address, then the object syllables resulting from it will always be left-justified in the object program. NABUR will, if necessary, generate a sufficient number of "dummy" NOP instructions (zero operator syllables) as filler. On the other hand, if a source line is not assigned a symbolic address,

TABLE 1  
 Octal Correspondences

Label	Octal Equivalent
RESERVED	snag 400000
	indi 200000
	bar 000055
	bpr 000054
	sar 000060
	iar 000063
	pcr 000057
	rtc 000114
	ipr 000110
	tfc 000124
	xir 000122
	idr 000074
	isr 000040
	pdr 000064
	rcr 000062
	rpr 000044
	ssr 000050
	ccr 000123
	rir 000130
	psr1 000100
	psr2 000102
	rpr1 000047
	stk1 000140
	stk2 000144
	stk3 000150
	stk4 000154
	pu 000001
	mu 000002
	xu 000003
	pr 000005
	mr 000006
xr 000007	
SEMIRESERVED	inbar no fixed value; initially zero
	inbpr no fixed value; initially zero
	insar no fixed value; initially zero

then the object syllables resulting from it will not necessarily start out a new computer word. This means that data specified by either a FLO or an INT pseudo-instruction will not necessarily be placed in a single word. The four syllables of data that result will "go around the corner," if necessary. It is important to remember that each instruction to which a jump is made must be assigned a symbolic address.

It has proven advantageous to give the thin-film registers their common names, so that the assembly routine can fill in their actual addresses where required. This has been accomplished in a straightforward manner by permanently reserving these names and addresses in the table of octal correspondences. Table 1 lists the labels currently accorded such preferential treatment by the NABUR system.

The majority of these reservations connect the common names of D825 thin-film registers with their octal codes or actual addresses. The remainder relate binary numbers with mnemonics useful in coding variant syllables for the transmit modifier instruction (TRM), or simplify setting "snag" or "indirect address" bits in second or higher level, 16-bit absolute addresses. All of these correspondences are fixed. The semire-served labels, on the other hand, correspond to no fixed octal codes or actual addresses. Thus, they are reserved in name only. For example, in the DEF (Define) pseudo-instruction they tell the assembly routine what the D825 program being assembled expects to have "in BAR," "in BPR," or "in SAR."

A label must not contain more than five characters; if it does, an error statement is printed out and only the first five characters enter the table of octal correspondences. The total number of distinct labels that may be used by the programmer must not exceed 2747. If this rule is violated, another error printout will result.

#### 4. REGULAR OR QUASI-REGULAR D825 INSTRUCTIONS

NABUR language is patterned after the machine language of the D825 computer, as defined

NOTE: NABUR rules, as distinguished from NAR rules, permit only one type of symbolic address, namely, the internal symbolic address. In particular, there are no numbered instructions or external symbolic addresses in the NABUR language. Nevertheless, table look-up and table entry are accomplished by means of subroutines written for the NAREC Assembly Routine. The authors wish to thank Alan B. Bligh for making these subroutines available to them.

in the Instruction Index (Appendix D of Ref. 2). In fact, it might be said to follow D825 language to the letter, because it upholds both the mnemonics for instruction names and the syllable layouts. Instructions listed in Appendix A form the class of "regular instructions" of the D825; that is, they are all machine-language instructions. Some of these regular instructions involve variant syllables that are particularly difficult to specify. For convenience in these cases, the mnemonic for a particular octal order number has been extended to form a set of related mnemonics that carry extra information in the operation symbol. This move obviates much programmer tedium. Such related mnemonics are called "quasi-regular instructions." In every case the use of quasi-regulars in the NABUR system is recommended but not mandatory.

First of all, regular operations STF (store thin film) and LTF (load thin film) each have two-member quasi-regular sets; containing respectively, STF (store thin fragment), STW (store thin word); and LTF (load thin fragment), LTW (load thin word). Secondly, the regular operation XLC (Index, Limit-Compare) has been split into an eight-member set containing XNN (Index, None), XEQ (Index, Equal), XGR (Index, Greater than), XGE (Index, Greater than or Equal to), XLS (Index, Less than), XLE (Index, Less than or Equal to), XUE (Index, Unequal), and XUC (Index, Unconditional). Thirdly, the regular operation SHF (Shift) has been split into an eight-member set containing DDA (Drop-Off, Double, Arithmetic), DDL (Drop-Off, Double, Logical), DSA (Drop-Off, Single, Arithmetic), DSL (Drop-Off, Single, Logical), EDA (End-Around, Double, Arithmetic), EDL (End-Around, Double, Logical), ESA (End-Around, Single, Arithmetic), and ESL (End-Around, Single, Logical). Notice that these shift quasi-regulars can be differentiated from the remaining operation symbols by the fact that none of the latter start with either a "D" or an "E."

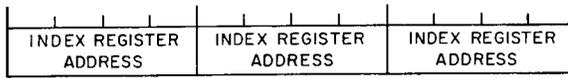
#### 5. TRIADIC FORMS

The following syllables must be written as triadic forms: (a) X-INDEX, (b) Iv-INDEX INCREMENT, (c) F-FIELD DEFINITION, and (d) Ri-REPEAT INCREMENT. This means that they are to be treated as three successive address quantities, some of which may be understood, in writing



## 7. INDEX SYLLABLES

### X-Index Syllable



Index syllables are expressed as triadic forms (see Part I, Section 5) and appear following the quantity or quantities they modify, in contrast to their ultimate position, as syllables, in the object program. All quantities  $Q_i$  designating index registers, must satisfy the inequality  $0 \leq Q_i \leq 15$ .

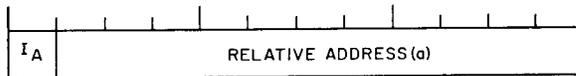
The three 4-bit portions of the X-index syllable, which correspond to the information given at the right of the first slash in an indexed address grouping, are filled directly if the index registers are nonsymbolically stated. But if the index registers are symbolically stated, these portions are filled from the label table. Suggested symbolic labels to be given index registers are X1, X2, ... For example, an expression like

$$+ \text{sam}/1/x3$$

would mean "index the address corresponding to sam by means of index register 1 and the index register preassigned to or defined as x3."

## 8. BASIC ADDRESS SYLLABLES (MEMORY AND BRANCH)

### M-Memory Address Syllable



If a memory syllable refers to an indirect address, the corresponding address grouping should bear a "-" prefix; otherwise, it should bear a "+" prefix (which may be understood). An unindexed memory syllable should be written as an address grouping of the form:

$$\pm AI$$

where **AI** is a simple address quantity,  $Q_1$ . As one binary digit of the memory syllable is allotted

to the indirect-address bit, the quantity  $Q_1$  has to satisfy the inequality

$$0 \leq Q_1 \leq 2047.$$

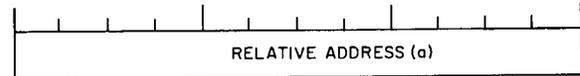
An indexed memory syllable must be written as an address grouping **A** having some one of the following forms:

$$\begin{aligned} &Q_1/Q_2/Q_3/Q_4 \\ &Q_1/Q_2/Q_3 \\ &Q_1/Q_2 \end{aligned}$$

where  $Q_1$ ,  $Q_2$ ,  $Q_3$ , and  $Q_4$  are address quantities. Moreover,  $Q_1$  designates a relative address and  $Q_2$ ,  $Q_3$ , and  $Q_4$  designate index registers. In all cases the indexing quantities follow the basic quantities in an indexed situation. This has been done for the programmer's benefit, despite the fact that the D825 object program has its syllables stored the other way around. Nor should it be overlooked that a hidden correction is applied in each memory-syllable determination. The result put in the object program will be " $Q_1$  minus the contents of the BAR" or " $Q_1$ -inbar." The programmer can override this at his own option by including "inbar" explicitly within the quantity. That is, he must write " $Q_1$ +inbar."

It is often useful to be able to bring the negative value of inbar, the current contents of the BAR, to an index register. At first glance the rules of NABUR will not permit this. Still, it can be done with only slight inconvenience by taking  $Q_1$  to be "+0-inbar."

### B-Branch Address Syllable

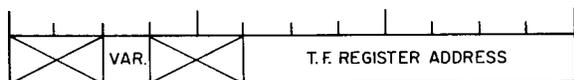


The branch-address syllable differs from the memory syllable in that it does not have an indirect address bit position. The forms are the same. Consequently, the value of  $Q_1$  may have its maximum range, *i.e.*,  $0 \leq Q_1 \leq 4095$ . Again, note that a hidden correction is applied in each branch-address syllable determination. The result inserted in the object program will be " $Q_1$  minus the contents of the BPR" or " $Q_1$ -inbpr." The

programmer can cancel this by the device employed in the preceding paragraph. That is, he must write "Q1+inbpr."

### 9. SPECIAL-USE SYLLABLES

#### T-Thin-Film Address Syllable



In the NABUR system, the D825 thin-film registers follow their octal codes given in the 16-bit T.F. Register Map on page C-2 and the 12-bit T.F. Register Map on page C-3 of Ref. 2. The programmer, accordingly, may write octal integers from "000," (or simply "0,") to "177," to refer to the thin-film registers. Alternatively, he may write the corresponding decimal equivalents "0" to "127" or use either the common names already reserved in advance in the table of octal correspondences (Table 1) or new names, defined or preassigned by pseudo-instructions, such as x1, x2, etc., for index registers. But he may not number the limit registers in decimal, as shown on page C-2, Ref. 2 (*i.e.*, "0" through "15"), because this conflicts with the earlier sequence for index registers. He could, of course, label the limit registers r0, ..., r15, and to these labels preassign or define decimal values 16 through 31. Note however that decimal values "0" through "15" and octal values "0" through "17" must be used when designating limit registers in the Iv-Index Increment Variant Syllable.

The association of octal codes to the labels of thin-film registers is complicated by a restriction to key registers (marked by heavy dots in the T.F. maps on pages C-2 and C-3 of Ref. 2) when an entire 48-bit word is being referenced. Failure to select a key register to head a multiregister store (STW) or load (LTW) thin-film quasi-regular instruction leads to an error statement, unless the address quantity involved is indexed, and therefore of unverifiable correctness as far as the assembler is concerned.

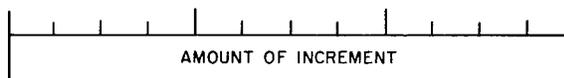
#### Iv-Index Increment Variant Syllable



This syllable is used only with the XLC (Index Limit Compare) instruction. Treatment of the last two portions of the syllable has already been described. The variant consists of a bit for "increase" or "decrease" and three bits to determine the branch condition.

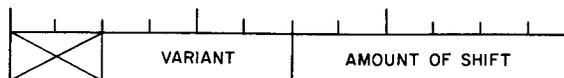
The "+" or "-" prefixed to the Index Amount Syllable leads to a "0" or "1" placed in the Index Increment Variant Syllable. The XLC instruction has been split into eight quasi-regular instructions, as shown in Appendix C, IX, and the particular one used will determine the contents of the remaining three bits in the variant.

#### Ia-Index Increment Amount Syllable



The programmer must prefix a "+" or "-" character to the Ia syllable to indicate increment and decrement, respectively. This will lead to placement of a "0" or "1" in the proper binary digit of the Iv-Index Variant Syllable. The value of the Ia syllable quantity has the range  $0 \leq Q \leq 4095$ .

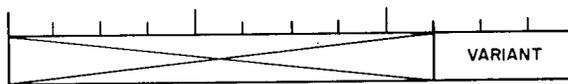
#### S-Shift Syllable



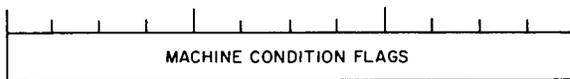
The programmer must prefix a "+" or "-" character to the shift syllable address grouping to denote the direction of shifting. The interpretation of this sign will agree with the conception of a right shift as a multiplication by a negative power of two (*i.e.*, minus exponent implies use of "-" prefix) and of a left shift as a multiplication by a positive power of two (*i.e.*, plus exponent implies use of "+" prefix). This convention circumvents a possible need for introducing four-letter operation symbols.

The prefix will contribute one bit to the variant. The shift syllable has been split into eight quasi-regular operations, as described in Appendix C, IX, and the use of one of these will determine the contents of the remaining three variant bits. The shift amount Q has the following range:

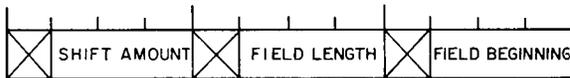
$$0 \leq Q \leq 63.$$

**Vt-Transmit Variant Syllable**

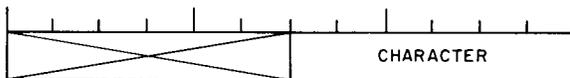
The variant in this syllable is contributed by the program through translation of various reserved labels (pu, mu, xu, pr, mr, xr) which appear in the associated address grouping. Meanings are attached to these labels in Appendix C, X.

**L-Logical Machine Condition Syllable**

The Logical Machine Condition syllable will be treated as a SYL-type syllable.

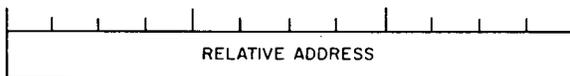
**F-Field Definition Syllable**

This is a triadic syllable. The associated address quantities must lie in the range  $0 \leq Q \leq 7$ .

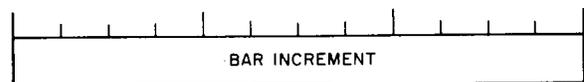
**C-Character Syllable**

The 6-bit character of the character syllable is treated as a SYL-type syllable. The range of the quantity  $Q$  is as follows:

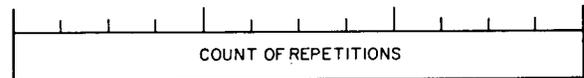
$$0 \leq Q \leq 63.$$

**Ja-Subroutine Jump Address Syllable**

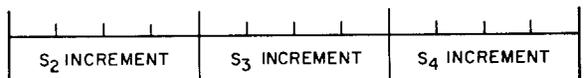
The subroutine jump address syllable will be treated as a SYL-type syllable, except that once more a useful, but concealed, correction is applied in each determination. The function of this correction is to make more natural the use of the machine's abilities. The resulting Ja syllable, inserted in the object program, will be "Q1 minus the contents of the SAR" or "Q1-insar." If it is ever necessary to do so, the programmer can override this correction by writing "Q1+insar."

**Ji-Subroutine Jump Increment Syllable**

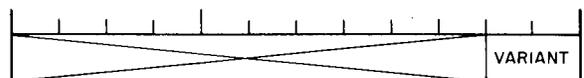
The subroutine jump increment syllable will be treated as a SYL-type syllable. Its determination can be made relative to the BAR by writing "Q1-inbar."

**Rc-Repeat Count Syllable**

The repeat count syllable will be treated as a SYL-type syllable.

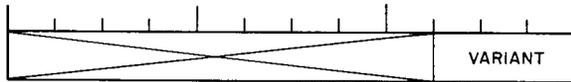
**Ri-Repeat Increment Syllable**

The programmer should rely on Ref. 2 for occasional use of the repeat instruction in order that he may be able to conform to the machine-language-instruction format. This syllable is processed as a triadic form.

**IO-I/O Syllable**

The range of  $Q$  for the I/O syllable is  $0 \leq Q \leq 3$ .

### Vs-Special Register and Computer Interrupt Variant Syllable



The range of  $Q$  for the special register and computer interrupt variant syllable is  $0 \leq Q \leq 7$ .

## 10. PSEUDO-INSTRUCTIONS

The following paragraphs describe pseudo-instructions currently available to programmers using the NABUR system. A pseudo-instruction which will handle I/O Descriptors is under consideration.

### INT (Integer)

This pseudo-instruction is used to insert a signed, 47- binary-digit integer as four successive syllables in the object program. The programmer must ensure that this integer arrives in the object program as a single D825 word, if so required. This always can be done by marking the NABUR word containing the INT instruction with a symbolic address. For clarity, since only one address grouping is required, the programmer may allow the integer quantity to spill over into more than one address-grouping field. And he may use spaces in a free fashion unless he is using a compound address quantity, for this would terminate incorrectly at the first encountered space. Although this operation was designed explicitly for signed and unsigned decimal or octal integers in the range  $-2^{47}$  to  $2^{47}$ , it is possible to use a label instead of an integer or to use concatenation symbols to tack on additional address units, such as labels.

### EXAMPLES

(How to set the contents of "spot" equal to the numerical address corresponding to "dot.")

spot INT dot

(How to set up a second or higher-level 16-bit absolute address with indirect or snag bits.)

INT 65535+indi  
INT 65535+snag  
INT 65535+snag+indi

### FLO (Floating Point)

This pseudo-instruction converts the number which follows the operation symbol into standard D825 floating-point binary form. It requires all three address groupings for its specification, so the rule for spaces is not relaxed in this case, as it was for INT. The FLO operation can handle only signed or unsigned decimal numbers. The mantissa will be normalized, and any overflow or underflow that occurs in the exponent will be detected and considered an error. True zero will be converted into standard D825 floating-point-zero form (i.e., 7777 0000 0000 0000).

In the FLO pseudo-instruction, a number is defined by the expression

$$(A1 \times 10^{A2}) \times 2^{A3}$$

where **A1**, **A2**, and **A3** refer to the three consecutive address groupings, reading from left to right. The required form of these three distinctly treated address groupings is as follows. The **A1** form will contain (a) from 1 to 18 decimal characters, (b) a leading plus or minus, the plus being optional, and (c) a decimal point, also optional, but if missing assumed to be located at the immediate right of the last digit. The form of **A2** and **A3** will be that of any address quantity, such as is allowed in the SYL pseudo-instruction.

### SYL (Syllable)

This pseudo-instruction translates the quantity in the first address grouping, **A1**, of a NABUR word from source language into a twelve-digit binary integer and places this result in the syllable that is next to be filled in the object program. Explaining this another way, it puts the syllable in the position indicated by the  $Q$ -arrow. (See Part II, Section 6.) The form of the data in a SYL address grouping is identical to that found in an ordinary, unindexed branch syllable address grouping. In other words, it is an address quantity.

The SYL pseudo-instruction injects an arbitrary syllable explicitly into the object program. For

example, it is the only direct way to place a non-zero NOP operator syllable (*i.e.*, one having a 1 in some address indicator bit position) in the object program. SYL also can be used to place a HLT (Halt) operator syllable in the object program, whose six address indicator bit positions contain an identifying integer.

### PRE (Preassign)

The table of octal correspondences may be expanded by the source program in three separate ways. (In the unlikely event that this table becomes full, the assembly process will record an error statement.) One way, described earlier, occurs when a new symbolic address is encountered by the assembly process sometime during the first pass. The actual address, in octal, that corresponds to the symbolic address so detected is a function of the present position of the Q-arrow. Occasionally, however, the programmer may wish to preassign an octal address to a label. For example, if a programmer using NABUR wishes to utilize, with no further assembly, object tapes occupying known positions in the D825 core memory, he can preassign symbolic addresses to correspond to their actual machine addresses.\* A particular label can be preassigned or assigned only once during any assembly run; however, it can be defined more than once, or even assigned (or preassigned) a value initially, and then defined to be some other value, later. Moreover, several different labels can be given the same value, either by preassignment, assignment, definition, or some combination of these, if required.

### DEF (Define)

The DEF pseudo-instruction is another way in which to introduce values into the table of octal correspondences. (The remaining way is by assigning a symbolic address to a NABUR word; this is discussed elsewhere.) The DEF operation is similar to the PRE operation. But with PRE, only new labels can be entered into the table of octal

\*In the NABUR system, preassignment is accomplished by a pseudo-instruction that is treated in the same manner as any other NABUR instruction, and consequently preassign tapes and source tapes are conceptually identical, or may even be made into one tape. Nevertheless, despite the fact that NABUR preassignments can be intermingled with the steps of the source program, it is advisable to write all the preassigns necessary for an assembly on one tape, to be read in before the other tapes.

correspondences. Should there already be an entry having the same spelling, the program will accept the new value, but also will provide an error statement. On the other hand, with DEF, it is possible to give new octal values to symbolic addresses or labels that have already been entered in the table, provided only that they are unreserved. So, in contrast to the PRE pseudo-instruction, the programmer may change the value corresponding to any unreserved or semireserved label. For example, he may change those values corresponding to "inbar, inbpr, or insar" at points where he expects the running program to reset the BAR, BPR, or SAR registers. The same rules for address groupings that hold for PRE also apply to DEF. All labels will be stored in the table of octal correspondences on the first pass; any label which will not appear in this table at the end of the first pass must necessarily be an error. Define operations, but not preassigns or assigns, are reiterated on the second pass.

Clearly, D825 programs must reset the BAR at least once every 2048 words. The assembly routine must be told what setting applies at various points to compensate the values from the table of octal correspondences.

### RES (Reserve)

The reserve pseudo-instruction sets aside a stated number of consecutive syllable positions located next in the object program leaving them with zero contents. Only one address grouping is used, which contains an unindexed address quantity of SYL form. This is acted upon by the Adar subroutine to establish the number of syllable positions that are supposed to be kept vacant.

### NOR (Normal Mode)

This pseudo-instruction requires no address groupings for its specification. Its purpose is to signal that further operation of the D825 program being assembled is to be in the normal mode.

### CON (Control Mode)

This pseudo-instruction is the reverse of NOR. Its purpose is to signal that further operation of the D825 program being assembled is to be in the control mode. It also requires no address groupings.

NABUR OPERATOR INSTRUCTION SHEET (6/11/62)      Date \_\_\_\_\_

RCC Problem Number \_\_\_\_\_      NRL Account Number \_\_\_\_\_

Problem Title \_\_\_\_\_      Programmer \_\_\_\_\_

Assembly \_\_\_\_\_      Telephone \_\_\_\_\_

1. Format: Hexadecimal, Infinite Column. Output: Line Printer. (LO 12C1).

2. First Address in Object Program 0000000 \_\_\_\_\_.

3. Source Program Tapes (Place on slow reader before heading):

	Tape Label	Check Sum	Cont. P.B.		Tape Label	Check Sum	Cont. P.B.
(1)	_____	_____	"	(9)	_____	_____	"
(2)	_____	_____	"	(10)	_____	_____	"
(3)	_____	_____	"	(11)	_____	_____	"
(4)	_____	_____	"	(12)	_____	_____	"
(5)	_____	_____	"	(13)	_____	_____	"
(6)	_____	_____	"	(14)	_____	_____	"
(7)	_____	_____	"	(15)	_____	_____	"
(8)	_____	_____	"	(16)	_____	_____	"

4. (RO 12C2). Stops at 1390 if no errors.

5. If there are assembly errors:    STOP    or    PROCEED.

6. If no assembly errors: Change output to: Line Printer and Paper Tape Punch.  
Push Continue.

7. Label output tape \_\_\_\_\_

8. Special Instructions:

Fig. 4 - NABUR operator instruction sheet

## 11. NABUR OPERATOR INSTRUCTION SHEET

Figure 4 shows a NABUR Operator Instruction Sheet. It is more or less self-explanatory. Notice that "Format" is already specified. It will be necessary for the programmer, however, to decide upon the intended first address of the object program in the D825 memory. This address must be entered in the space provided on line 2. It is possible to write as many as 16 source tape labels on one sheet, along with the check sums for these tapes. To write more than 16, simply attach another sheet by stapling, and renumber the appropriate lines of the second sheet, starting with (17), (18), and so on. Also, make a note under line 8 (Special Instructions) telling the machine operator to continue reading tapes in the order indicated on the next sheet.

## 12. SOURCE TAPES

The steps followed in preparing NABUR source tapes are similar to those followed for NAR source tapes. Both kinds of tapes are read into the NAREC and therefore must have a related structure; namely, they must contain a heading of the standard form preceded by heading recognition symbols, and the body of the source program must be preceded and followed by the standard NAREC prepare and end routines, respectively. All characters in the body of the source program, excluding tape feeds and deletes, will enter the NAREC memory, including all format characters (CR, tab, and space). Recognition of format characters is an intrinsic part of the assembly process.

But NABUR source tapes do differ from NAR source tapes in many ways. For example, both a title and other expository matter may be included within the body of a NABUR tape by treating this material as a commentary. Comments, it may be recalled, (a) are preceded by a "CR tab" combination, (b) are typed, and (c) do not reappear as part of the checking output. To obtain something, called a remark, which does reappear, it is necessary to place an "r space" combination immediately after the tab following the last address grouping.

More than one source tape can be read into the NAREC before starting the NABUR assembly process; thus, for example, several subroutines can be read in before or after the main program. In this case, the object program will be assembled

from the component tapes in the consecutive order of their reading. In the event of a faulty reading, it will be possible for the machine operators to reread only the last source tape in an attempt to obtain the required check sum. In planning several tapes for the same assembly, remember that a given symbolic address can be assigned only once, whether this be by assignment (writing a label in the symbolic address field), or by pre-assignment. The NABUR system will record an error statement whenever this rule is violated. The total number of characters permitted for the subject data is 32,768; the total number of words allowed for the object program is 4096. Longer programs could be handled with some minor changes to the assembly routine.

## 13. ERROR STATEMENTS

Considerable checking is built into the NABUR system, and an appropriate error statement (Table 3) is printed out whenever an error is detected. Although no hard and fast rule is possible, it is usually the case that when a source mistake is detected, an erroneous syllable will be formed, partially correct, partially zero. Such faulty syllables are not easily corrected by over-punching, because the object tape is coded in biocital; nevertheless, they often can be corrected at the D825 console.

TABLE 3  
NABUR System Error Statements

Number	Message
1	number or label too large
* 2	illegal op code
3	the iar cannot be loaded in normal mode
4	symbol table capacity exceeded
* 5	duplicate label
6	improper stack reference
7	missing operand
8	index identifier not previously defined
* 9	index assigned not between <u>0</u> and <u>15</u>
*10	index improperly designated
11	improper amount syllable
12	xlc inst improper index or limit register identifier
13	indirect branch
*14	improper octal number
15	improper character field definition
16	single shift amount greater than <u>47</u>
17	improper shift specification

TABLE 3 (Continued)  
NABUR System Error Statements

Number	Message
18	double shift amount greater than <u>11</u>
19	invalid numbering of an instruction
20	illegal input to octal digit function
21	halt used in control mode
*22	faulty symbolic address
23	illegal branch condition designated
24	label not previously defined
25	argument not previously defined
26	i-o instruction used in normal mode
27	missing bus designation
28	improper special register designation
29	lsr used in normal mode
30	improper value preassigned or defined
31	improper t.f.r. key address
32	improper t.f.r. address
33	missing operator
34	concatenation element not previously defined
35	repeat increment greater than <u>15</u>
36	item vector exceeded
*37	label reserved
*38	improper number of spaces in specification field
39	illegal character definition
40	improper indirect address
41	syllable greater than <u>4095</u>
42	undefined base register label
43	improper number on tape record
44	relative address greater than <u>2047</u>
*45	improper decimal number
46	argument not defined
*47	mistake in source line

\*Most commonly seen.

## Part II THE NABUR ROUTINE

### 1. NAME OF PROGRAM

NABUR - A NAREC Assembler for the Burroughs D825 Modular Data-Processing Computer

### 2. CLASS

L1 *Executive Routines, Assembly*  
O2 *Simulation and/or Interpretation of Other Computers, Other*

### 3. PURPOSE

a. To assemble an object program in Burroughs D825 machine language from one or more programs written in NABUR source language.

b. To check for programming errors and print out corresponding error statements wherever appropriate.

c. To provide, in the event of an error-free assembly, both a punched object tape that is compatible with the requirements of the D825 input format and a dual-language hard copy (or "checking output") in which source lines and object words appear collaterally.

### 4. LANGUAGE USED

NAR 1B and NAREC interpretive language

### 5. AUTHORS

R. M. Mason and I. G. Fishman

### 6. FORMAL STATEMENT

When the NABUR program has been placed in its working position in the NAREC, and a start order executed, the program will clear portions of the memory, reset certain parameters, and almost immediately call for a keyboard insertion by the NAREC machine operator of the intended first address for the D825 object program to be produced, *i.e.*, the octal integer written by the programmer in the space provided on line 2 of the NABUR Operator Instruction Sheet. The machine operators will box this first address as a right-justified hexadecimally coded octal integer and push the transfer button, whereupon the program will convert it to pure octal and store the result in qadd, a working location in the NABUR program. Next the program will call for the source tapes to be read in, including format characters but excluding delete and tape feed codes, as a succession of 6-bit characters. The heading is read in as well, and then printed out, but not stored permanently. The (augmented) check sum for each component source tape is computed and recorded. After all source tapes for the assembly have been read in, forming the subject data, a second start order will begin the assembly process.

Consider Fig. 5. Upon readin, the material in Fig. 5a takes on the appearance shown in Fig. 5b. After readin of all the source tapes to be assembled, the NABUR program begins its first pass

through the subject data. In both the first and second passes a subroutine called "Subject Delineator" is employed to distinguish in memory the current line of subject data to be examined. It does not place this line in a detached situation. In Fig. 5b a "P-arrow" applied successively at points P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>, and P<sub>4</sub> denotes the first character of each string to be considered. Also, a "Q-arrow" might be defined that always points to the next available syllable position in the object program.

A line of subject data is defined as a non-empty string of characters other than carriage returns, preceded by, but not including, one or more CR's on the left (open), and followed by one or more CR's, but including only the first (closed), on the right. In Fig. 5b, the 18 characters at P<sub>1</sub>, namely 0 through CR, would comprise the first line of subject data to be isolated by the Subject Delineator subroutine. The NABUR program treats each source line independently, using the character handling and manipulating programs of the CHAMP system. An examination of what would happen with the most general form of line will illustrate practically the whole approach. The most general form of source line is:

```
X TAB SYM* TAB OPS SPACE A1 SPACE A2 SPACE A3 TAB R* CR
```

where

X is a line designation

SYM is a symbolic address

OPS is a three-letter mnemonic operation symbol

A1, A2, and A3 are address groupings  
which may contain indexing

R is "r space remark"

\* denotes that the starred element is optional.

The positions of tabs and spaces are critical in identifying particular components of the source line. In the first pass, only a framework of the object program resulting from assembly is created. The number of tabs per line must always total three, and if this rule is broken, the assembly process will record an error statement to that effect on the first pass. The character substring enclosed by the first two tabs is taken to be the symbolic address. If there is no entry between the first two tabs, the assembly program accepts the fact that this line is not marked, and goes on. If there is an entry, tests are made of the acceptability of the symbolic address, following the rules given earlier in this report (Part I, Section 3).

If acceptable, the symbolic address is next compared with the entries in the table of octal correspondences. If it does not duplicate any entry of the table, whether it be reserved, semireserved, or unreserved, it is entered along with the current address of the object program, qadd. The presence of a symbolic address on a line will reset qadd to the left syllable position, starting a new line if necessary. Should the symbolic address fail any of the tests made on it, suitable error statements will result, and usually, faulty syllables can be expected to appear in the object program.

Next to be examined is the operation symbol. It is always present in every line of subject data containing a NABUR word and consists of three letters—no more, no less—between the second tab and the first space.

If the operator symbol is that of a regular or quasi-regular machine instruction, a subroutine called "Oper" is brought into play. First, this subroutine recognizes the machine operation involved and begins to fill the operator syllable by inserting the corresponding order number. Then, depending on the type of instruction, it searches the subject data to find the accompanying address groupings in the source line and sees whether the required number of groupings is present, whether the memory syllables are allowed to be stack references, etc. This address indicator information is "OR"ed into the operation syllable, by means of a "Variant Bit" subroutine, called "Vabi." (Appendix B has block diagrams of the "Oper" and "Vabi" subroutines.) Once the operator syllable is filled, it is inserted into the object frame. During the first pass, room that will be required during the second pass for non-operator syllables is filled in temporarily with zeros to maintain the correct structure of the object program. Pseudo-instructions are handled differently from regular and quasi-regular machine instructions. They do not contribute an operator syllable. Some may transmit a signal as to the mode of operation, normal or control; others may generate matter that contributes one, or four, syllables to the object program. Where the pseudo-instruction does contribute syllables (usually data) to the object program, sufficient zero syllables will be placed in the object structure in the first pass.

It may be worthwhile to discuss how syllables are stored in the object program. From one to four syllables, once assembled, whether by the "Oper" subroutine, by other subroutines in the second

pass, or by generation of zeros, are placed in a NABUR working location, called "qsyl". Since four syllables comprise one 48-bit D825 computer word, there are 12 bits or four octal digits per syllable. A subroutine called "Store Q" can be invoked to store one, two, three, or four syllables from qsyl into the current position in the object program, as indicated by the Q-arrow. The first syllable may be placed in any of the four possible positions it can occupy in a computer word.

This is the extent of the first pass. Incidental checking has been taking place as to the validity of the operation code, the correct number and kind of address groupings required for the particular operation, and format in general, all with corresponding error statements. After the first pass, the object frame for the preceding example (Fig. 5) would be as shown in Fig. 6 (remember that all digits are octal in the object program).

0		stf bpr n
1	arcb	ltf n bar
00		def inbpr arcb
01		def inbar arcb
2		stw srr n
3		ltw n +74,
4		bad h n n
5		ltf n 17,
6		esl n +1 n
7		brb n n arcb2
8	arcbl	trm n pu n
9		cls h arcc1 arcb3
10		cgr h arcc2 arcb4
11		bmu h n n
12		bsu n arcc3 n
13		dsa n +2 n
14		bmu h h n
15		bsu arck5 h n
16		bdv arck4 n n
17		bad n arck3 n

Fig. 5A - Passage of source text

		Character number							
		1	2	3	4	5	6	7	8
Word number 1	CR	0 ← P <sub>1</sub>	TAB	a	r	c	b	TAB	
2	s	t	f	SPACE	b	p	r	SPACE	
3	n	TAB	CR	1 ← P <sub>2</sub>	TAB	TAB	l	t	
4	f	SPACE	n	SPACE	b	a	r	TAB	
5	CR	0 ← P <sub>3</sub>	0	TAB	TAB	d	e	f	
6	SPACE	i	n	b	p	r	SPACE	a	
7	r	c	b	TAB	CR	0 ← P <sub>4</sub>	l	TAB	
8	TAB	d	e	f	SPACE	i	n	b	
9	a	r	SPACE	a	r	c	b	TAB	
10	CR	2	TAB	TAB	s	t	w	SPACE	
11	s	s	r	SPACE	n	TAB	CR	3	
12	TAB	TAB	l	t	w	SPACE	n	SPACE	
13	+	7	4	,	TAB	CR	4	TAB	
14	TAB	b	a	d	SPACE	h	SPACE	n	
15	SPACE	n	TAB	CR	5	TAB	TAB	l	
16	t	f	SPACE	n	SPACE	l	7	,	

Fig. 5B - Corresponding source tape image

Fig. 5 - Readin example

		Character number							
		1	2	3	4	5	6	7	8
Word number 17	TAB	CR	6	TAB	TAB	e	s	l	
18	SPACE	n	SPACE	+	l	SPACE	n	TAB	
19	CR	7	TAB	TAB	b	r	b	SPACE	
20	n	SPACE	n	SPACE	a	r	c	b	
21	2	TAB	CR	8	TAB	a	r	c	
22	b	l	TAB	t	r	m	SPACE	n	
23	SPACE	p	u	SPACE	n	TAB	CR	9	
24	TAB	TAB	c	l	s	SPACE	h	SPACE	
25	a	r	c	c	l	SPACE	a	r	
26	c	b	3	TAB	CR	l	0	TAB	
27	TAB	c	g	r	SPACE	h	SPACE	a	
28	r	c	c	2	SPACE	a	r	c	
29	b	4	TAB	CR	l	l	TAB	TAB	
30	b	m	u	SPACE	h	SPACE	n	SPACE	
31	n	TAB	CR	l	2	TAB	TAB	b	
32	s	u	SPACE	n	SPACE	a	r	c	
33	c	3	SPACE	n	TAB	CR	l	3	
34	TAB	TAB	d	s	a	SPACE	n	SPACE	
35	+	2	SPACE	n	TAB	CR	l	4	
36	TAB	TAB	b	m	u	SPACE	h	SPACE	
37	h	SPACE	n	TAB	CR	l	5	TAB	
38	TAB	b	s	u	SPACE	a	r	c	
39	K	5	SPACE	h	SPACE	n	TAB	CR	
40	l	6	TAB	TAB	b	d	v	SPACE	
41	a	r	c	K	4	SPACE	n	SPACE	
42	n	TAB	CR	l	7	TAB	TAB	b	
43	a	d	SPACE	n	SPACE	a	r	c	
44	K	3	SPACE	n	TAB	CR			

Fig. 5B (Continued) – Corresponding source tape image

Fig. 5 (Continued) – Readin example

STF 1540	(BPR) 0000	LTF 3010	(BAR) 0000
STW 1540	(SSR) 0000	LTW 3010	(+ 74) 0000
BAD 6520	LTF 3010	(17,) 0000	ESL 3610
(+ 1) 0000	BRB 2602	(ARCB2) 0000	0000
TRM 3410	(PU) 0000	CLS 7432	(ARCC1) 0000
(ARCB3) 0000	CGR 7532	(ARCC2) 0000	(ARCB4) 0000
BMU 6120	BSU 6410	(ARCC3) 0000	DSA 3610
(+2) 0000	BMU 6124	BSU 6444	(ARCK5) 0000
BDV 6040	(ARCK4) 0000	BAD 6510	(ARCK3) 0000

Fig. 6 - Object frame

The first zero location remains to be filled in during the second pass with the address of BPR read from the label table. The n, denoting normal stack reference, does not call for a syllable; but rather, is coded in the appropriate address indicator bits of the operator syllable. The assembly routine automatically takes care of this conversion for the programmer. After going through all the source lines, the assembly routine enters its midpass phase and prints out the contents of the table of octal correspondences. Then it decides whether or not to go on to the second pass. At the time of printout, any errors considered to be sufficiently damaging to the program as a whole will result in setting a "whoa" flag. This flag later will attempt to terminate the program at midpass, after the assembly routine prints out the table of octal correspondences followed by five asterisks.

The second pass contributes the rest of the syllables to the object program structure, does extensive checking of the address groupings themselves, and prints out a checking output

for each line as it is formed. The appearance of this output will be described in detail later (see Part II, Section 8).

As in the first pass, each line of subject data is considered independently, and again use is made of the Subject Delineator subroutine. This time, however, the operator syllable is passed over, and direct attention is given to the address groupings accompanying the operation symbol. Regular machine operations are matched to a routing table which sequences control through the different types of action blocks needed to completely specify syllables for the operation. For example, the first line of subject data previously examined uses operation STF. This STF corresponds to octal order number "15," which is added to a constant to produce a jump address that sends the control to a routing table. Control, under the direction of this table, is then sent through the memory syllable subroutine and returns to "home." In the third subject line of the same example, the routing table for BRB will send control to the memory syllable subroutine, again to the memory syllable subroutine, then to the branch syllable subroutine, and finally to "home." Each action block will check for its particular needs in the address grouping and will form the corresponding object syllable and store it in the object program. Special action blocks are written for indexed syllables, triads, etc. Inside an address grouping, address arithmetic is permitted for certain syllables, notably the memory syllable. The address units may be symbolic addresses, or unsigned decimal or octal numbers, as long as they conform to the rules of writing NABUR language described in the first part of this report. Newly constructed syllables are stored in the waiting empty spaces in the object program. After examining the line of subject data in question, and after returning from the routing table indicated by the operation, the NABUR system produces a line of checking output.

Moreover, in the second pass, lines containing pseudo-instructions are processed somewhat differently. In the case of CON (control mode), NOR (normal mode), DEF (define), and PRE (preassign), the appropriate action is taken, setting addresses in the symbolic address table, or flags to indicate mode, but no syllables are added to the object program. In the case of RES (reserve), the required number of syllables is again set equal to zero.

In the case of FLO, INT, and SYL, the number of syllables entered into the object program is four, four, or one, respectively. After each insertion the corresponding line of checking output is produced. Since they are pseudo-instructions, FLO, INT, and SYL never directly contribute operator syllables.

The above procedure is carried out for each line of subject data. When all such lines have been exhausted, another determination is made by the program as to whether any error printouts have occurred in the second pass. If at least one error has been detected, the program will stop immediately. If no errors have been found, the program will produce a D825 punched paper tape containing the bioctal object program. This will be reproduced on the line printer also, but since the characters punched are bioctal, and the line printer is hexadecimal, it will be unintelligible, and will serve only as notice that an output octal tape has been produced.

## 7. OPERATOR INSTRUCTIONS

The NABUR Operator Instruction Sheet has been discussed in Part I of this report. Current operator instructions for loading the NABUR program are posted in the NAREC machine room.\*

## 8. OUTPUT

The assembly routine starts printout (Fig. 7a) by providing a heading for the output: three carriage returns, followed by "NABUR D825 MODULAR DATA-PROCESSING SYSTEM ASSEMBLY," followed by four carriage returns. Next, a heading "READIN INFORMATION" is supplied, followed by two more carriage returns. The assembly routine also furnishes two types of information in connection with every source tape read in. First the word "HEADING-" is printed and is immediately followed by the heading of the source tape. This heading can include the tape label, title, name of programmer, date, etc. If a particular source tape has no heading, the word "NONE" will be printed. The second type of information furnished is the (augmented) check sum of the body of the source

tape. This helps when a particular source tape is required for subsequent assembly. If the check sum has been entered on the NABUR operator instruction sheet next to the tape label, accurate reading of the source tape may be certified. The above printout occurs before and during reading of tapes. The only other printouts possible during the first pass are error statements. They have the same three-line form whether they occur during the first or second pass, as follows.

```

XXXXXXXXXXXXX
          n      mmmm
          sssssssssssssssssssssssss

```

The x's stand for the hexadecimal contents of the location in the assembly program from which the error message was generated; n stands for the error tally, which is an ascending decimal counter beginning with one; the m's stand for the error message itself; and the s's stand for the particular line in the source program in which the error was detected.

Whether or not errors have been detected in the first pass, the program goes into midpass (Fig. 7b). At this time, the heading "SYMBOLIC ADDRESSES USED" is printed out, followed by a table. This table always begins with the reserved addresses, starting at "snag" and ending with "xr," and the semireserved addresses, "inbar" through "insar." Next will follow, in order, the labels found in the first pass. The printout of the table of octal correspondences will have the form:

```

oooooo      aaaaaa

```

where the o's represent the 18-bit octal integer associated with a label, and the a's represent the 1-to-5 character label. Upon completion of this printout, the assembly routine decides whether or not to go on to the second pass. If at least one error has been recorded in the first pass, the assembler will print out five asterisks and come to a stop. Otherwise, the assembler will start the second pass (Fig. 7c). First, the column headings "ADDR," "OBJECT SYLLABLES," "LINE NO," "SYMBOL," "OP," "SOURCE LINE," and "REMARKS" are printed to clarify the checking output that will come a segment at a time after each source line has been examined. (Recall that in

\*Copies of these directives are available to qualified persons upon request. Address inquiries to Director, U. S. Naval Research Laboratory, Washington, D. C., 20390, Attention Code 4550.

the second pass, the nonoperator syllables are placed in the object frame, which was constructed during the first pass.)

The checking output has the following form:

```
nnnn mmmmmm | ssss ssss ssss ssss | kkkk aaaaa ops A1 A2 A3
```

where

nnnn stands for natural numbers generated by the assembler to number the output lines decimally in ascending order

mmmmm stands for the actual octal address of the object line

ssss represents the object program syllables, consisting of four octal digits each (if an octal syllable does not apply, four dots are printed instead)

[uprights] enclose the four object syllables  
kkkk stands for the line designation found on the source tape

aaaaa represents the symbolic address or label associated with the source line, if any

ops denotes the three-letter NABUR operation symbol

**A1, A2, and A3** stand for the associated address groupings, which may or may not be indexed, and may or may not all be present, depending upon the particular instruction used.

NABUR output is illustrated by the following sample, Fig. 7, which has resulted from assembling together four source tapes. (A portion of the handwritten programming for the first source tape is reproduced in Fig. 2. The complete typed copy of this source tape appears as Fig. 3.) The checking output is designed for programmer convenience in studying the octal D825 machine-language program that is assembled. To this end, the output page has been divided more or less down the middle, the left half applying to the object program, and the right half, to the source program. In theory, a line of source data should correspond to a line of object data. In practice, however, the source data may require several lines to be printed within the limits of its column. No new line of object data is provided until the source line corresponding to the preceding line of object data has been completely printed. Similarly, no new source-line data are provided until all the object line has been printed, within the limits of its column. Since an instruction in the D825 computer may require from one to seven syllables, there is no actual instruction word, as

such. The format of the object-syllable field relates the object line to the particular source line. It is split into four syllable columns, each with positions for four octal digits. Bona fide zero syllables contain four octal zeros. Each pertinent object syllable is printed out in its actual location in the object tape. Therefore, it so happens that if all the lines in the column headed "object syllables" were compressed upward so that a line with dots in a syllable position would be filled with a following line's syllable, all dots would disappear, no object syllables would be lost, and a true representation of the object tape would result.

The presence of an r in the remark indication field of the NABUR programming sheet will cause the following remark, and the r itself, to be typed by the typist and to be printed out in the checking output. The remark is printed out on a separate line after all object material and corresponding subject material for the particular line have been printed. The only other type of printout which may occur during generation of the checking output is the error printout, which is described earlier in this section. Should no errors be found in second pass, the assembly program will provide output in the form of both a punched paper tape and line-printed material. The tape and hard copy contain the assembled actual bioctal object program for the D825 computer (Fig. 7d).

## 9. TAPE LABELS

Tapes with basic tape number 4825 are used; current versions are available to the machine operators.

## 10. ADDRESS INFORMATION

Not pertinent.

## 11. EXTERNAL WORKING MEMORY

Not applicable.

## 12. VERSATILITY

The system is self-contained and does not lend itself to programmer changes.

## 13. REMARKS

The assembler at present requires about ten seconds to process each source line.

### ACKNOWLEDGMENTS

The authors wish to thank Harold Ashby and Frank Zurcher of the Burroughs Laboratories at Paoli, Pennsylvania, authors of the Burroughs 220/D825 Modular Processor Assembler, for permitting them to study this program thoroughly before work on NABUR began. The authors also appreciate the willingness of the Burroughs staff to explain why certain features of their approach to the problem are imposed by the logical design of the D825 and of its operating system, *i.e.*, the AOSP (1,3).

The authors would also like to acknowledge the most cooperative help given by Alan B. Bligh,

Bruce Wald, and Alice Jo Campbell in setting up the NABUR conventions.

### REFERENCES

1. Anderson, J.P., Hoffman, S.A., Shifman, J., and Williams, R.J., "D825 - A Multiple Computer System for Command and Control," pp. 86-96, in *AFIPS Conference Proc.*, Vol. 22, 1962 Fall Joint Computer Conference, Spartan:Washington, D. C., 1962
2. "The Burroughs D825 Modular Data Processing System, Programming Manual," Burroughs Corporation, Contract Nonr 3521(00)(x), Jan. 31, 1962
3. Thompson, R.N., and Wilkinson, J.A., "The D825 Automatic Operating and Scheduling Program," in *AFIPS Conference Proc.*, Vol. 23, 1963 Spring Joint Computer Conference, Spartan:Baltimore, 1963

UNCLASSIFIED

```

NNNN
0000
INPUT 0001 DIRECTOR TABLE 10CNS. 3683-368C
INPUT 0001 ORDER LINE VECTOR TABLE 10CNS. 36F3-36F3
TRANSFER CONTROL TO SYSTEM-START AT ORDER LINE 0001

SYSTEM TAPE INPUT- HEADING--
4825-0000 NABUR 1 RMASON/RS 8/11/62

FIRST WORD 0000- ED27 00 8019 61
LAST WORD 091F- 0000 00 0000 00
STANDARD CHECK SUM EA6C 6B 396A 58
AUGMENTED CHECK SUM EA6C A4 2211 58

SYSTEM TAPE INPUT- HEADING--
4825-0920 NABUR 1 FISHMAN/MM 12-5-62
1-8-63ZJ
1-16-63ZJ
1/18/63MM
3-28-63ZJ
FIRST WORD 0920- 093F 42 0942 42
LAST WORD 0D78- 9556 40 0000 00
STANDARD CHECK SUM 5EE9 6F 03F8 AF
AUGMENTED CHECK SUM 5EE9 DF 61DF AF

SYSTEM TAPE INPUT- HEADING--
NONE
FIRST WORD 0D80- 0D9E 42 0D9F 43
LAST WORD 1545- 0000 00 0000 00
STANDARD CHECK SUM 532E DD 4D47 D0
AUGMENTED CHECK SUM 532F 85 9ED6 D0

NAREC INSTR. EXECUTED BY SYSTEM 12C1 10

NNNN
0000

NABUR D-825 MODULAR DATA-PROCESSING SYSTEM ASSEMBLY

READIN INFORMATION

HEADING-
4815 ARCSIN/ARCCOS SUBROUTINE CAMPBE11/ZJ 1-31-63
3-29-63ZJ

CHECK SUM 5D391B995709

HEADING-
4817 SIN SUBROUTINE CAMPBE11/MM 2/1/63
    
```

Problem number	396
Operator	W.T.
Checked by	
Time	2145

Fig. 7A - Portion obtained during readin

Fig. 7 - Sample output (printout only). This figure represents the concluding step for the example given in Figs. 2, 3, 5, and 6.

FORM 801 PRINTED BY THE PHOTODUPLICATION COMPANY, U.S.A.

```

●      3-4-63ZJ
● CHECK SUM 2FB23F3AE9C9
HEADING-
● 4818  COSINE SUBROUTINE  CAMPBE11/EJ  2-1-63
  2/13/63MM
●      3-4-63ZJ
● CHECK SUM 96C5B960514A
● HEADING-  S4R19  SQUARE ROOT SUBROUTINE  CAMPBE11/EJ  2/12/63RS
●
●      3-4-63ZJ
● CHECK SUM 2E9A18C07A2A
HEADING-
● 4820  BINARY TO DECIMAL OUTPUT ROUTINE  CAMPBE11  2/1/63MM
  3-12-63ZJ
● CHECK SUM 2B83022B7522

```

Fig. 7A (Continued)—Portion obtained during readin

Fig. 7 (Continued) — Sample output (printout only). This figure represents the concluding step for the example given in Figs. 2, 3, 5, and 6.

```

● NNNN

● 0000
  SYMBOLIC ADDRESSES USED

● 400000 SNAG
  200000 INDI
  000055 BAR
● 000054 BPR
  000060 SAR
  000063 IAR
● 000057 PCR
  000114 RTC
  000110 IPR
  000124 TFC
● 000122 XIR
  000074 IDR
  000040 ISR
  000064 PDR
  000062 RCR
● 000044 RPR
  000050 SSR
  000123 CCR
  000130 RIR
● 000100 PSR1
  000102 PSR2
● 000047 RPR1
  000140 STK1
  000144 STK2
● 000150 STK3
  000154 STK4
● 000001 PU
  000002 MU
  000003 XU
  000005 PR
● 000006 MR
  000007 XR
  000000 ---
● 000214 INBAR
  000214 INBPR
  000000 INSAR
● 000000 ARCB
  000004 ARCB1
  000014 ARCB2
● 000015 ARCB3
  000021 ARCB4
  000024 ARCB1
● 000031 ARCB6
  000032 ARCB7
  000033 AB71
  000034 AB72
● 000035 AB73
  000036 ARCC1
  000037 ARCC2
● 000040 ARCC3
  000041 ARCK5
  000042 ARCK4
● 000043 ARCK3
  000044 ARCK2
  000045 ARCK1
● 000046 ARKK2
  000047 ARKK1
● 000050 ARCC4

```

Fig. 7B — Portion obtained at midpass

Fig. 7 (Continued) — Sample output (printout only). This figure represents the concluding step for the example given in Figs. 2, 3, 5, and 6.

● 000051 ARCC5  
 ● 000052 ARCKA  
 ● 000053 ARCKB  
 ● 000054 ARCC6  
 ● 000055 SINBG  
 ● 000061 SINB1  
 ● 000063 SINB2  
 ● 000066 SINB3  
 ● 000073 SINB4  
 ● 000074 SINB5  
 ● 000075 SINB6  
 ● 000103 SMU1T  
 ● 000104 SHA1F  
 ● 000105 SØNE  
 ● 000106 SPART  
 ● 000107 SINK4  
 ● 000110 SINK3  
 ● 000111 SINK1  
 ● 000112 SINC9  
 ● 000113 SINC7  
 ● 000114 SINC5  
 ● 000115 SINC3  
 ● 000116 SINC1  
 ● 000117 SP1  
 ● 000120 CØBEG  
 ● 000124 CØB1  
 ● 000126 CØB2  
 ● 000127 CØB3  
 ● 000134 CØB4  
 ● 000135 CØB5  
 ● 000136 CØB6  
 ● 000140 CØB7  
 ● 000146 CØC1  
 ● 000147 CØC2  
 ● 000150 CØC3  
 ● 000151 CØC4  
 ● 000152 CØC5  
 ● 000153 CØK6  
 ● 000154 CØK4  
 ● 000155 CØK2  
 ● 000156 CØK0  
 ● 000157 CØK9  
 ● 000160 CØK7  
 ● 000161 CØK5  
 ● 000162 CØK3  
 ● 000163 CØK1  
 ● 000164 SQBEG  
 ● 000167 SQR1  
 ● 000173 SQB2  
 ● 000176 SQB3  
 ● 000201 SQB4  
 ● 000202 SQB5  
 ● 000204 SQC1  
 ● 000205 SQC2  
 ● 000206 SQK12  
 ● 000207 SQK11  
 ● 000210 SQK10  
 ● 000211 SQKK2  
 ● 000212 SQKK1  
 ● 000213 SQKK0  
 ● 000214 ØUTPR  
 ● 000221 ØBØ  
 ● 000223 ØB1  
 ● 000230 ØB2  
 ● 000231 ØB3

FORM 801 PRINTED BY THE STANDARD REGISTERS COMPANY, U.S.A.

Fig. 7B (Continued) — Portion obtained at midpass

Fig. 7 (Continued) — Sample output (printout only). This figure represents the concluding step for the example given in Figs. 2, 3, 5, and 6.



	ADDR	OBJECT	SYLLABLES	LINE NO	SYMBOL	OP	SOURCE	LINE	REMARKS
0	000000	11540	0054	.....	10	ARCB	STF	BPR	N
1	000000	.....	.....	3010	0055		1TF	N	BAR
2	000001						DEF	INBPR	ARCB
3	000001						DEF	INBAR	ARCB
4	000001	11540	1050	.....	12		STW	SSR	N
5	000001	.....	.....	3010	1074		1TW	N	+74,
6	000002	16520	.....	.....	14		BAD	H	N N
7	000002	.....	3010	0017	.....		1TF	N	17,
8	000002	.....	.....	.....	3610		ES1	N	+1 N
9	000003	.....	2602	0014	.....		BRB	N	N ARCB2
10	000003	.....	.....	.....	0000	ARCB1	TRM	N	PU N
11	000004	.....	.....	7432	0036		C1S	H	ARCC1 ARCB3
12	000005	.....	7532	0037	0021		CGR	H	ARCC2 ARCB4
13	000006	16120	.....	.....	.....		BMU	H	N N
14	000006	.....	6410	0040	.....		BSU	N	ARCC3 N
15	000006	.....	.....	.....	3610		DSA	N	+2 N
16	000007	.....	6124	.....	.....		BMU	H	H N
17	000007	.....	.....	6444	0041		BSU	ARCK5	H N
18	000010	16040	0042	.....	.....		BDV	ARCK4	N N
19	000010	.....	.....	6510	0043		BAD	N	ARCK3 N
20	000011	16400	.....	.....	.....		BSU	N	N N
21	000011	.....	6040	0044	.....		BDV	ARCK2	N N
22	000011	.....	.....	.....	6510		BAD	N	ARCK1 N
23	000012	.....	6100	.....	.....		BMU	N	N N
24	000012	.....	.....	6510	0050		BAD	N	ARCC4 N
25	000013	12240	0031	.....	.....		UCT	ARCB6	
26	000013	.....	.....	0000	0000	ARCB2	XUC	+1	//17, /0, ARCB1
27	000015	16124	.....	.....	.....	ARCB3	BMU	H	H N
28	000015	.....	6444	0041	.....		BSU	ARCK5	H N
29	000015	.....	.....	.....	6040		BDV	ARCK4	N N
30	000016	.....	6510	0043	.....		BAD	N	ARCK3 N
31	000016	.....	.....	.....	6400		BSU	N	N N
32	000017	16040	0046	.....	.....		BDV	ARCK2	N N
33	000017	.....	.....	6510	0047		BAD	N	ARCK1 N
34	000020	16100	.....	.....	.....		BMU	N	N N
35	000020	.....	2240	0031	.....		UCT	ARCB6	
36	000020	.....	.....	.....	0000	ARCB4	STF	17,	N
37	000021	.....	.....	3010	0037		1TF	N	37,
38	000022	13030	1160	.....	.....		1TW	H	+160,
39	000022	.....	.....	6440	0051		BSU	ARCC5	N N
40	000023	11450	0003	0000	.....		SRJ	3,	0,
41	000023	.....	.....	.....	0000	ARCR1	STF	37,	N
42	000024	.....	.....	3010	0017		1TF	N	17,
43	000025	11540	1160	.....	.....		STW	+160,	N
44	000025	.....	.....	6110	0053		BMU	N	ARCKB N
45	000026	16440	0052	.....	.....		BSU	ARCKA	N N
46	000026	.....	.....	6100	.....		BMU	N	N N
47	000026	.....	.....	.....	6440		BSU	ARCC6	N N
48	000027	.....	1540	1074	.....		STW	+74,	N
49	000027	.....	.....	.....	3010		1TW	N	SSR

PRINTED BY THE STA

FORM 801

THE STANDARD REGISTER COMPANY, N. S. A.

Fig. 7C — Portion obtained during second pass — the checking output

Fig. 7 (Continued) — Sample output (printout only). This figure represents the concluding step for the example given in Figs. 2, 3, 5, and 6.

50	000030	1....	0000	0000	0000	148	ARCB6	UCT ARCB7/17,
		12260	0017	0032	....			
51	000031	1....	....	....	0000	149	ARCB7	SRR
		10400	....	....	....			
52	000032	1....	0000	0000	0000	150	AB71	TRM N XU N
		13410	0003	....	....			
53	000033	1....	....	0400	....	151		SRR
54	000033	1....	....	....	0000	152	AB72	BSU ARCC6 N N
		16440	0054	....	....			
55	000034	1....	....	0400	....	153		SRR
56	000034	1....	....	....	0000	154	AB73	BAD ARCC6 N N
		16540	0054	....	....			
57	000035	1....	....	0400	....	155		SRR
58	000035	1....	....	....	0000	156	ARCC1	INT 1324047463163255,
		11324	0474	6316	3255			
59	000037	1662	0367	4746	1000	157	ARCC2	INT 1662036747461000,
60	000040	10400	0000	0000	0000	158	ARCC3	INT 0400000000000000,
61	000041	11324	6525	5632	1603	159	ARCK5	INT 1324652556321603,
62	000042	14052	4102	6062	4515	160	ARCK4	INT -0052410260624515
63	000043	13435	3200	0065	6622	161	ARCK3	INT 3435320000656622,
64	000044	10312	0162	1751	2156	162	ARCK2	INT 0312016217512156,
65	000045	10414	6310	2013	1753	163	ARCK1	INT 0414631020131753,
66	000046	10624	0344	3722	4334	164	ARCK2	INT 0624034437224334,
67	000047	11031	4620	4026	3726	165	ARCK1	INT 1031462040263726,
68	000050	10622	0773	4244	3343	166	ARCC4	INT 0622077342443343,
69	000051	12000	0000	0000	0000	167	ARCC5	INT 2000000000000000,
70	000052	12127	1630	4511	6426	168	ARCKA	INT 2127163045116426,
71	000053	10256	4151	5104	4031	169	ARCKB	INT 0256415151044031,
72	000054	11444	1766	5212	2212	170	ARCC6	INT 1444176652122212,
73	000055	11540	0054	....	....	171	SINBG	STF BPR N
74	000055	1....	....	3010	0055	172		1TF N BAR
75	000056	1....	....	....	....	173		DEF INBPR SINBG
76	000056	1....	....	....	....	174		DEF INBAR SINBG
77	000056	10200	....	....	....	175		SSU
78	000056	1....	2602	0004	....	176		BRB N N SINB1
79	000056	1....	....	....	0300	177		SSD
80	000057	13630	0302	....	....	178		DS1 H +2 N
81	000057	1....	....	5400	....	179		1XR N N N
82	000057	1....	....	....	3610	180		ES1 N +1 N
		10201	....	....	....			
83	000060	1....	2240	0006	....	181		UCT SINB2
84	000060	1....	....	....	0000	182	SINB1	SSD
		10300	....	....	....			
85	000061	1....	6130	0026	....	183		BMU H SMU1T N
86	000061	1....	....	....	3610	184		DS1 N +10 N
		10312	....	....	....			
87	000062	1....	5400	....	....	185		1XR N N N
88	000062	1....	....	3610	0201	186		ES1 N +1 N
89	000063	10300	....	....	....	187	SINB2	SSD
90	000063	1....	3410	0001	....	188		TRM N PU N
91	000063	1....	....	....	3610	189		DSA N -2 N
		10502	....	....	....			
92	000064	1....	7432	0027	0011	190		C1S H SHA1F SINB3
93	000065	16440	0030	....	....	191		BSU SONE N N
94	000065	1....	....	2240	0011	192		UCT SINB3
95	000066	17532	0031	0020	....	193	SINB3	CGR H SPART SINB6
96	000066	1....	....	....	6110	194		BMU N SP1 N
		10042	....	....	....			
97	000067	1....	6124	....	....	195		BMU H H N
98	000067	1....	....	3630	0102	196		DSA H +2 N
99	000070	16510	0032	....	....	197		BAD N SINK4 N
100	000070	1....	....	6040	0033	198		BDV SINK3 N N
101	000071	16510	0034	....	....	199		BAD N SINK1 N

Fig. 7C. (Continued)—Portion obtained during second pass—the checking output

Fig. 7 (Continued) — Sample output (printout only). This figure represents the concluding step for the example given in Figs. 2, 3, 5, and 6.

102	000071	.....	6500	....	127	BAD	N	N	N
103	000071	.....	.....	6100	128	BMU	N	N	N
104	000072	3610	0111	....	129	DSA	N	+9	N
105	000072	.....	.....	0200	130	SSU			
106	000072	.....	.....	0000	131	SINB4	BRB	N	N
		2602	0017	....					
107	000073	.....	.....	0300	132				
108	000073	.....	.....	0400	133	SINB5	SSD		
109	000074	0300	.....	.....	134				
110	000074	.....	3410	0003	135		TRM	N	XU
111	000074	.....	.....	.....	136	SINB6	SRR		
112	000075	3610	0102	....	137		DSA	N	+2
113	000075	.....	.....	6124	138		BMU	H	H
114	000075	.....	.....	6130	139		BMU	H	SINC9
		0035	.....	.....					
115	000076	.....	6510	0036	140		BAD	N	SINC7
116	000076	.....	.....	6104	141		BMU	N	H
117	000077	6510	0037	....	142		BAD	N	SINC5
118	000077	.....	.....	6104	143		BMU	N	H
119	000077	.....	.....	6510	144		BAD	N	SINC3
		0040	.....	.....					
120	000100	.....	6100	.....	145		BMU	N	N
121	000100	.....	.....	6510	0041	146	BAD	N	SINC1
122	000101	6100	.....	.....	147		BMU	N	N
123	000101	.....	3610	0110	148		DSA	N	+8
124	000101	.....	.....	.....	0200	149	SSU		
125	000102	2240	0016	....	150		UCT	SINB4	
126	000102	.....	.....	0000	0000	151	SMU1T	INT	0013301330133013.
		0013	3013	3013	3013				
127	000104	0400	0000	0000	0000	152	SHA1F	INT	0400000000000000.
128	000105	1000	0000	0000	0000	153	SONE	INT	1000000000000000.
129	000106	0060	7107	2072	1575	154	SPART	INT	0060710720721575.
130	000107	3100	7574	2541	2540	155	SINK4	INT	3100757425412540.
131	000110	1011	3551	7137	2633	156	SINK3	INT	1011355171372633.
132	000111	5173	0434	7701	7655	157	SINK1	INT	-1173043477017655
133	000112	0000	2346	5735	1052	158	SINC9	INT	0000236657351052.
134	000113	4002	3111	4614	4267	159	SINC7	INT	-0002311146144267
135	000114	0012	1464	2573	1260	160	SINC5	INT	0012146425731260.
136	000115	4024	5273	6016	1302	161	SINC3	INT	-0024527360161302
137	000116	0014	4417	6651	0101	162	SINC1	INT	0014441766510101.
138	000117	3110	3755	2421	1033	163	SP1	INT	3110375524211033.
139	000120	1540	0054	.....	.....	164	COREG	STF	BPR
140	000120	.....	.....	3010	0055	165		ITF	N
141	000121					166		BAR	
142	000121					167		DEF	INBPR
143	000121	3410	0001	....	....	168		DEF	INBAR
144	000121	.....	.....	0200	....	169		TRM	N
145	000121	.....	.....	.....	2602	170		SSU	
		0004	.....	.....	.....			BRB	N
146	000122	.....	0300	.....	.....	171		SSD	
147	000122	.....	.....	3610	0002	172		ESA	N
148	000123	3630	0512	....	....	173		DSA	H
149	000123	.....	.....	2240	0006	174		UCT	C0B2
150	000124	0300	.....	.....	.....	175	C0R1	SSD	
151	000124	.....	6110	0026	....	176		BMU	N
152	000124	.....	.....	.....	3610	177		ESA	N
		0445	.....	.....	.....				
153	000125	.....	3630	0512	....	178		DSA	H
154	000125	.....	.....	.....	0000	179	C0R2	CGR	N
		7512	0027	0016	....				

Fig. 7C (Continued)—Portion obtained during second pass—the checking output

Fig. 7 (Continued) — Sample output (printout only). This figure represents the concluding step for the example given in Figs. 2, 3, 5, and 6.

155	000126	.....	.....	.....	0300	14		SSD
156	000127	7532	0030	0020	.....	15	C0R3	CGR H C0C3 C0B7
157	000127	.....	.....	.....	3610	16		DSA N +11 N
		U113	.....	.....	.....			
158	000130	.....	6120	.....	.....	17		BMU H N N
159	000130	.....	.....	6130	0033	18		BMU H C0K6 N
160	000131	6510	0034	.....	.....	19		BAD N C0K4 N
161	000131	.....	.....	6104	.....	20		BMU N H N
162	000131	.....	.....	.....	6510	21		BAD N C0K2 N
		0035	.....	.....	.....			
163	000132	.....	6100	.....	.....	22		BMU N N N
164	000132	.....	.....	6510	0036	23		BAD N C0K0 N
165	000133	3610	0105	.....	.....	24		DSA N +5 N
166	000133	.....	.....	0200	.....	25		SSU
167	000133	.....	.....	.....	0000	26	C0R4	BRB N N C0B5
		2602	0015	.....	.....			
168	000134	.....	.....	0300	.....	27		SSD
169	000134	.....	.....	.....	0400	28		SRR
170	000135	0300	.....	.....	.....	29	C0R5	SSD
171	000135	.....	3410	0003	.....	30		TRM N XU N
172	000135	.....	.....	.....	0400	31		SRR
173	000136	5410	0032	.....	.....	32	C0R6	1XR N C0B5 N
174	000136	.....	.....	0300	.....	33		SSD
175	000136	.....	.....	.....	6440	34		BSU C0C4 N N
		0031	.....	.....	.....			
176	000137	.....	2240	0007	.....	35		UCT C0B3
177	000137	.....	.....	.....	0000	36	C0R7	BSU C0C2 N N
		6440	0027	.....	.....			
178	000140	.....	.....	3610	0112	37		DSA N +10 N
179	000141	6124	.....	.....	.....	38		BMU H H N
180	000141	.....	6130	0037	.....	39		BMU H C0K9 N
181	000141	.....	.....	.....	6510	40		BAD N C0K7 N
		0040	.....	.....	.....			
182	000142	.....	6104	.....	.....	41		BMU N H N
183	000142	.....	.....	6510	0041	42		BAD N C0K5 N
184	000143	6104	.....	.....	.....	43		BMU N H N
185	000143	.....	6510	0042	.....	44		BAD N C0K3 N
186	000143	.....	.....	.....	6100	45		BMU N N N
187	000144	6510	0043	.....	.....	46		BAD N C0K1 N
188	000144	.....	.....	6100	.....	47		BMU N N N
189	000144	.....	.....	.....	3610	48		DSA N +8 N
		0110	.....	.....	.....			
190	000145	.....	0200	.....	.....	49		SSU
191	000145	.....	.....	2240	0014	50		UCT C0B4
192	000146	0013	3013	3013	3013	51	C0C1	INT 0013301330133013
193	000147	0001	0000	0000	0000	52	C0C2	INT 0001000000000000
194	000150	0000	4000	0000	0000	53	C0C3	INT 0000400000000000
195	000151	0002	0000	0000	0000	54	C0C4	INT 0002000000000000
196	000152	0000	0000	0000	0001	55	C0C5	INT 1,
197	000153	4000	5162	7467	0055	56	C0K6	INT -0000516274670055
198	000154	0010	0727	5323	7527	57	C0K4	INT 0010072753237527
199	000155	4047	3647	1053	7555	58	C0K2	INT -0047364710537555
200	000156	0037	7777	7760	7573	59	C0K0	INT 0037777777607573
201	000157	0000	2366	5735	1052	60	C0K9	INT 0000236657351052
202	000160	4002	3111	4614	4267	61	C0K7	INT -0002311146144267
203	000161	0012	1464	2573	1261	62	C0K5	INT 0012146425731261
204	000162	4024	5273	6026	5317	63	C0K3	INT -0024527360265317
205	000163	0014	4417	6651	0101	64	C0K1	INT 0014441766510101
206	000164	1540	0054	.....	.....	0	SQREG	STF BPR N
207	000164	.....	.....	3010	0055	1		1TF N BAR

Fig. 7C (Continued) — Portion obtained during second pass — the checking output

Fig. 7 (Continued) — Sample output (printout only). This figure represents the concluding step for the example given in Figs. 2, 3, 5, and 6.

208	000165						100	DEF INBPR SQBEG
209	000165						101	DEF INBAR SQBEG
210	000165		2000	.....	.....	.....	12	C1A N
211	000165		.....	3030	0017	.....	13	1TF H 17,
212	000165		.....	.....	.....	7606	14	CEQ N H SQB4
			0015	.....	.....	.....		
213	000166		.....	3410	0001	.....	15	TRM N PU N
214	000166		.....	.....	.....	0000	16	SQB1 CGR H SQC1 SQB2
			7532	0020	0007	.....		
215	000167		.....	.....	.....	7432	17	C1S H SQC2 SQB5
			0021	0016	.....	.....		
216	000170		.....	.....	3610	0502	18	DSA N -2 N
217	000171		6511	0022	.....	.....	19	BAD N SQK12 H
218	000171		.....	.....	6040	0023	10	BDV SQK11 N N
219	000172		6440	0024	.....	.....	11	BSU SQK10 N N
220	000172		.....	.....	2240	0012	12	UCT SQB3
221	000173		.....	0502	.....	.....	13	SQB2 DSA N -2 N
222	000173		.....	.....	6511	0025	14	BAD N SQKK2 H
223	000174		6040	0026	.....	.....	15	BDV SQKK1 N N
224	000174		.....	.....	6440	0027	16	BSU SQKK0 N N
225	000175		3610	0101	.....	.....	17	DSA N +1 N
226	000175		.....	.....	0000	0000	18	SQB3 SSD
			0300	.....	.....	.....		
227	000176		.....	6001	.....	.....	19	BDV N N H
228	000176		.....	.....	3610	0501	20	DSA N -1, N
229	000177		0300	.....	.....	.....	21	SSD
230	000177		.....	3610	0501	.....	22	DSA N -1, N
231	000177		.....	.....	.....	6504	23	BAD N H N
232	000200		0300	.....	.....	.....	24	SSD
233	000200		.....	6004	.....	.....	25	BDV N H N
234	000200		.....	.....	6500	.....	26	BAD N N N
235	000200		.....	.....	.....	0000	27	SQB4 DSA N -0, /17, N
			3614	0017	0500	.....		
236	000201		.....	.....	.....	0400	28	SRR
237	000202		3610	0102	.....	.....	29	SQB5 DSA N +2 N
238	000202		.....	.....	1252	0001	30	XUC +1 //17, /0, SQB1
			3760	0003	.....	.....		
239	000203		.....	.....	0000	0000	31	SQC1 INT 2000000000000000.
			2000	0000	0000	0000		
240	000205		1000	0000	0000	0000	32	SQB2 INT 1000000000000000.
241	000206		1044	4445	3605	7615	33	SQK12 INT 1044444536057615.
242	000207		0665	0713	0071	7361	34	SQK11 INT 0665071300717361.
243	000210		3453	6760	4417	3463	35	SQK10 INT 3453676044173463.
244	000211		2111	1112	7413	7432	36	SQKK2 INT 2111111274137432.
245	000212		1152	1267	5656	6320	37	SQKK1 INT 1152126756566320.
246	000213		2422	1203	2322	3372	38	SQKK0 INT 2422120323223372.
247	000214		2000	.....	.....	.....	0	OUTPR C1A N
248	000214						100	DEF INBPR OUTPR
249	000214						101	DEF INBAR OUTPR
250	000214		.....	3030	1014	.....	1	1TW H 14,
251	000214		.....	.....	.....	3010	2	1TF N 13,
			0013	.....	.....	.....		
252	000215		.....	3070	0017	0054	3	1TF 011M/17, 35.
			0035	.....	.....	.....		
253	000216		.....	3010	0034	.....	4	1TF N 34,
254	000216		.....	.....	.....	3010	5	1TF N 33,
			0033	.....	.....	.....		
255	000217		.....	3030	0032	.....	6	1TF H 32,
256	000217		.....	.....	.....	7416	7	C1S N 0ERR/17, 000
			0017	0056	0005	.....		
257	000220		.....	.....	.....	0100	8	H1T
258	000221		3574	0017	0061	0017	9	000 TRS 0BYN/17, 0WB/17,
			0066	.....	.....	.....		

FORM 801 - PRINTED BY THE STANBARD REGISTER COMPANY, U.S.A.

Fig. 7C (Continued) - Portion obtained during second pass - the checking output

Fig. 7 (Continued) - Sample output (printout only). This figure represents the concluding step for the example given in Figs. 2, 3, 5, and 6.

259	000222	.....	2240	0007	.....	10		UCT 081	
260	000222	.....	.....	.....	0000	11	081	C1A N	
261	000223	.....	3010	1010	.....	12		1TW N +10,	
262	000223	.....	.....	.....	3070	13		1TF 00NE/17, 31,	
263	000224	.....	0017	0055	0031	.....		ES1 0+INBAR/14, +1, N	
264	000225	.....	0014	0000	0201	.....		BRB N N 082	
265	000226	.....	3560	0017	0062	16		TRS 0SPCE/17, N	
266	000227	.....	2240	0015	.....	17		UCT 083	
267	000227	.....	.....	.....	0000	0000	082	TRS 0MINS/17, N	
268	000230	.....	.....	.....	0000	19	083	AIF 0W1/16./17, /1./1	
		.....	4077	7017	0067	0015		,/0./15, 0W1/16./17,	
		.....	0420	7017	0067	.....			
269	000232	.....	.....	.....	1252	20		X1E 401, //15./15, 08	
		.....	0401	2735	0022	.....			
270	000233	.....	.....	.....	2000	21		C1A N	
271	000234	.....	3010	0015	.....	22		1TF N 15,	
272	000234	.....	.....	1252	0001	23		XUC 1, //16./0, 084	
		.....	3740	0022	.....	.....			
273	000235	.....	.....	0000	0000	24	084	XGR 1, //11./11, 086	
		.....	1252	0001	1231	0026			
274	000237	.....	0200	.....	.....	25		SSU	
275	000237	.....	3410	0001	.....	26		TRM N PU N	
276	000237	.....	.....	.....	0000	27	085	BMU N 0TEN/17, N	
		.....	6114	0017	0060	.....			
277	000240	.....	.....	.....	2240	28		UCT 083	
		.....	0015	.....	.....	.....			
278	000241	.....	.....	0000	0000	0000	29	086	STW TFC N
		.....	1540	1124	.....	.....			
279	000242	.....	.....	1252	0001	30		X1S 1, //12./12, 085	
		.....	2252	0024	.....	.....			
280	000243	.....	.....	1252	0001	31		XGE 1, //14./14, 0811	
		.....	1714	0046	.....	.....			
281	000244	.....	.....	3070	0017	32		1TF 0F0UR/17, 31,	
		.....	0057	0031	.....	.....			
282	000245	.....	.....	1252	0001	33		XGE 1, //13./13, 0810	
		.....	1673	0043	.....	.....			
283	000246	.....	.....	2000	.....	34		C1A N	
284	000246	.....	.....	.....	3010	35		1TF N 11,	
		.....	0011	.....	.....	.....			
285	000247	.....	3560	0017	0062	36		TRS 0SPCE/17, N	
286	000250	.....	4077	7017	0067	0015	08A	AIF 0W1/16./17, /1./1	
		.....	0420	7017	0067	.....		,/0./15, 0W1/16./17,	
287	000251	.....	.....	.....	1252	38		X1E 401, //15./15, 08	
		.....	0401	2735	0041	.....			
288	000252	.....	.....	.....	2000	39		C1A N	
289	000253	.....	3010	0015	.....	40		1TF N 15,	
290	000253	.....	.....	1252	0001	41		XUC 1, //16./0, 089	
		.....	3740	0041	.....	.....			
291	000254	.....	.....	0000	0000	42	089	X1S 1, //11./11, 088	
		.....	1252	0001	2231	0034			
292	000256	.....	2240	0007	.....	43		UCT 081	
293	000256	.....	.....	.....	0000	0000	44	0810	C1A N
		.....	2000	.....	.....	.....			
294	000257	.....	.....	3010	0013	.....	45	1TF N 13,	
295	000257	.....	.....	.....	3070	46		1TF 0F0UR/17, 11,	
		.....	0017	0057	0011	.....			
296	000260	.....	.....	.....	3560	47		TRS 0CR/17, N	
		.....	0017	0063	.....	.....			

FORM 801 PRINTED BY THE PHOTODUPLICATION COMPANY, WASHINGTON, D.C.

Fig. 7C (Continued) — Portion obtained during second pass — the checking output

Fig. 7 (Continued) — Sample output (printout only). This figure represents the concluding step for the example given in Figs. 2, 3, 5, and 6.

297	000261	....	....	2240	0034	48		UCT	000
298	000262	3560	0017	0063	....	49	0011	TRS	0CR/17, N
299	000262	....	....	....	4077	50		AIF	0W1/16,/17, /1,/1
		7017	0067	0015	0420	, /0,/15, 0W1/16,/17,			
		7017	0067	....	....				
300	000264	....	....	1252	0401	51		X1E	401, //15,/15, 00
		2735	0046	....	....	11			
301	000265	....	....	1252	0001	52		XNN	1, //16,/0, 0,
		0340	7564	....	....				
302	000266	....	....	3574	0017	53		TRS	0END/17, 0W1/16,/
		0065	7017	0067	....	17,			
303	000267	....	....	....	0100	54		H1T	
304	000270	0000	0000	0000	3407	55	011M	INT	3407,
305	000271	0000	0000	0000	0001	56	00NE	INT	1,
306	000272	0000	0000	0000	0017	57	0ERR	INT	17,
307	000273	0000	0000	0000	0004	58	0F0UR	INT	4,
308	000274	0000	0000	0000	0012	59	0TFN	INT	12,
309	000275	5353	4545	4545	5353	60	0BYN	INT	-1353454545455353
						,			
310	000276	0000	0000	0000	0060	61	0SPCE	INT	60,
311	000277	0000	0000	0000	0053	62	0CR	INT	53,
312	000300	0000	0000	0000	0040	63	0MINS	INT	40,
313	000301	4444	4444	4444	4444	64	0END	INT	-0444444444444444
						,			
314	000302	0000	0000	0000	0000	65	0W0	INT	0,
315	000303	0000	0000	0000	0000	66	0W1	INT	0,

NNNN

Fig. 7C (Continued)—Portion obtained during second pass—the checking output

Fig. 7 (Continued) — Sample output (printout only). This figure represents the concluding step for the example given in Figs. 2, 3, 5, and 6.

```

P('8H*P(H/8HHQ!08HVEH<T6<ICHTQAEP-AF1LOWH(EHT<L4W))+(.IH?W+(;IH
L!H/2(9+JTF+ L4W))+(.IH?W+(;IH:L2(9P(V8HF88)W( I/0P(F8HVP(I+LH!W(JLW(I*P
(HQ8HH/2+VA CH0 W(' I(' -4 Q)ZA*ZR0K;H -415SAZ0(J)<*R
PCDA!Y20+TRV 1S I)HO-V(N40)F2?CH9,0(6F6N2MA.JB?01728
IW6<S) (9I;VYJ+2+P('8H*6< 0E80<'EH<T2(N0L86EH0+'EH<T0CHTEH><0
A7|W(82(I+A90LH.L4E8T<IHA+(B!HC!LEHT|<6<V0 0CH0 EHT<L4L8D!HEL IHFL IH(LI
H)LEHTH<2(Z-8-8-8- H*#MOGP-9+Q5)5(HID #F6B A C=TX*3Y#DHJ(<9I,I.K+IW
5Δ+(4JA+Z-<I;VY TT9HF*41HBP('8H*CHT<6< 0EH<E8>+2(N0LH6EH
E8>+7Z0+A80EHT-L0LAB!HCL !HDL!HEHT><6<P0 0CH0 'HA0W(92(MW(7EHT+L4L8F!
H(L !H)L !H.L!H?LEHTH<2(I-8-8-8-T<T( RQK*HM7(3+7(:E:H(+*F=#+Δ3Y#DHJ
(<9I,I.K+IW5Δ+L(4JA*6!VI;VY TTP('8H*088VXNPCHT+A0MQA1ZEH><I|2+(3W(42
(+EH><!|5*(6W(7EHTT0+TEH>T0EH>T! 0+ !EIV> EHT<+JTF*00HH);
E>XPNIM-#ΔLC(K+;VC)1|+Q-QA|J+KSS)042+032B0088H18H-8UV'D8HC8HB88AQZVS>TD
QV|VY2(M08HHH8UV*9EUI<T6<ID+VR2(PD+VW(=UVKP OUVK+J T7D208HP+JTF(2+JT+96<
CHTLIV+2(PP(I4+JT2J4+JTVI,8UV#9+JTZΔ?08HID+VR(=UVKP OUVK+J T7D)08HP+JTF(
)+JT29C2(M08H-8UV#ID*V12(CD*V)(=UVKP OUVK+J T7D,+JT0(+WDQV!UVKTCMTV +!|

```

[[+[(;);););];

MMMMMM

MMMMMMMM

NNNN

801 PRINTED BY THE STANDARD REGISTER COMPANY, N.S.A.

Fig. 7D — Portion obtained at conclusion of assembly

Fig. 7 (Continued) — Sample output (printout only). This figure represents the concluding step for the example given in Figs. 2, 3, 5, and 6.

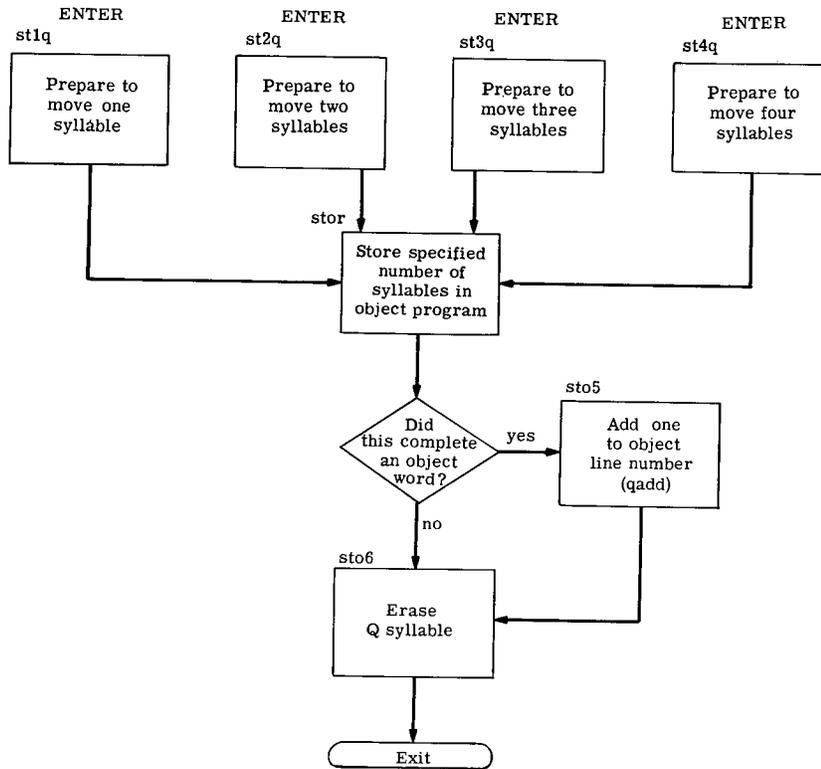
## Appendix A

### NUMERICAL LISTING OF BURROUGHS D825 INSTRUCTIONS

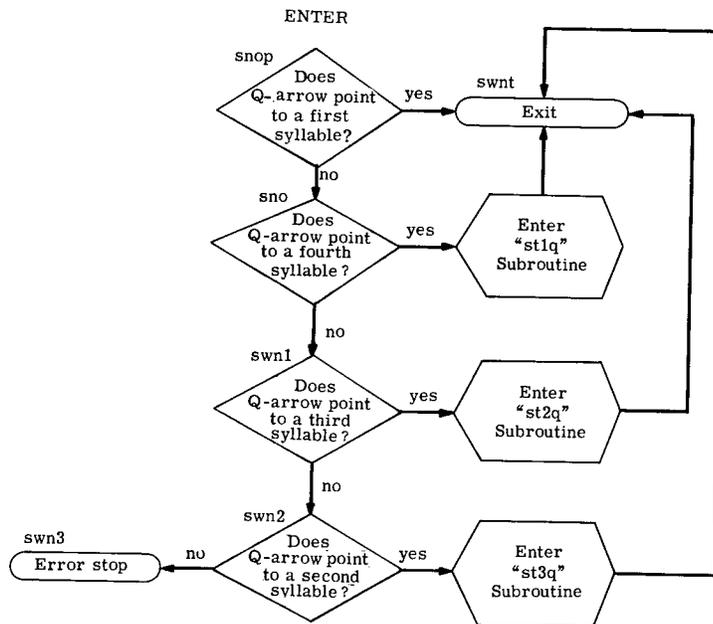
00	NOP	No Operation	40	AIF	Adjust and Insert, Field
01	HLT	Halt	41	SAF	Strip and Adjust, Field
02	SSU	Step Stack Up	42	BSF	Binary Subtract, Field
03	SSD	Step Stack Down	43	BAF	Binary Add, Field
04	SRR	Subroutine Return	44	LOF	Logical OR, Field
05	IRR	Interrupt Return	45	LXF	Logical EXCLUSIVE OR, Field
06	RVS	Reverse Stack	46	LCF	Logical COMPLEMENT, Field
07			47	LAF	Logical AND, Field
10	RPT	Repeat	50	CLF	Compare Less, Field
11	BRC	Branch on Condition	51	CGF	Compare Greater, Field
12	XLC	Index Limit Compare	52	CEF	Compare Equal, Field
13			53		
14	SRJ	Subroutine Jump	54	LXR	Logical EXCLUSIVE OR
15	STF	Store Thin Film	55	LOR	Logical OR
16	TIO	Transmit to Input/Output	56	LAN	Logical AND
17			57		
20	CLA	Clear	60	BDV	Binary Divide
21	SER	Store External Requests	61	BMU	Binary Multiply
22	UCT	Unconditional Transfer	62	FDV	Floating Divide
23			63	FMU	Floating Multiply
24	LCM	Logical COMPLEMENT	64	BSU	Binary Subtract
25	CBF	Convert Binary to Floating Point	65	BAD	Binary Add
26	BRB	Branch on Bit	66	FSU	Floating Subtract
27			67	FAD	Floating Add
30	LTF	Load Thin Film	70	ACL	Alphanumeric Compare Less
31	LSR	Load Special Register	71	ACG	Alphanumeric Compare Greater
32	CSE	Character Search	72	ACE	Alphanumeric Compare Equal
33			73		
34	TR	Transmit Modified	74	CLS	Compare Less
35	TRS	Transmit	75	CGR	Compare Greater
36	SHF	Shift	76	CEQ	Compare Equal
37			77		

**Appendix B**  
**BLOCK DIAGRAMS OF KEY SUBROUTINES**

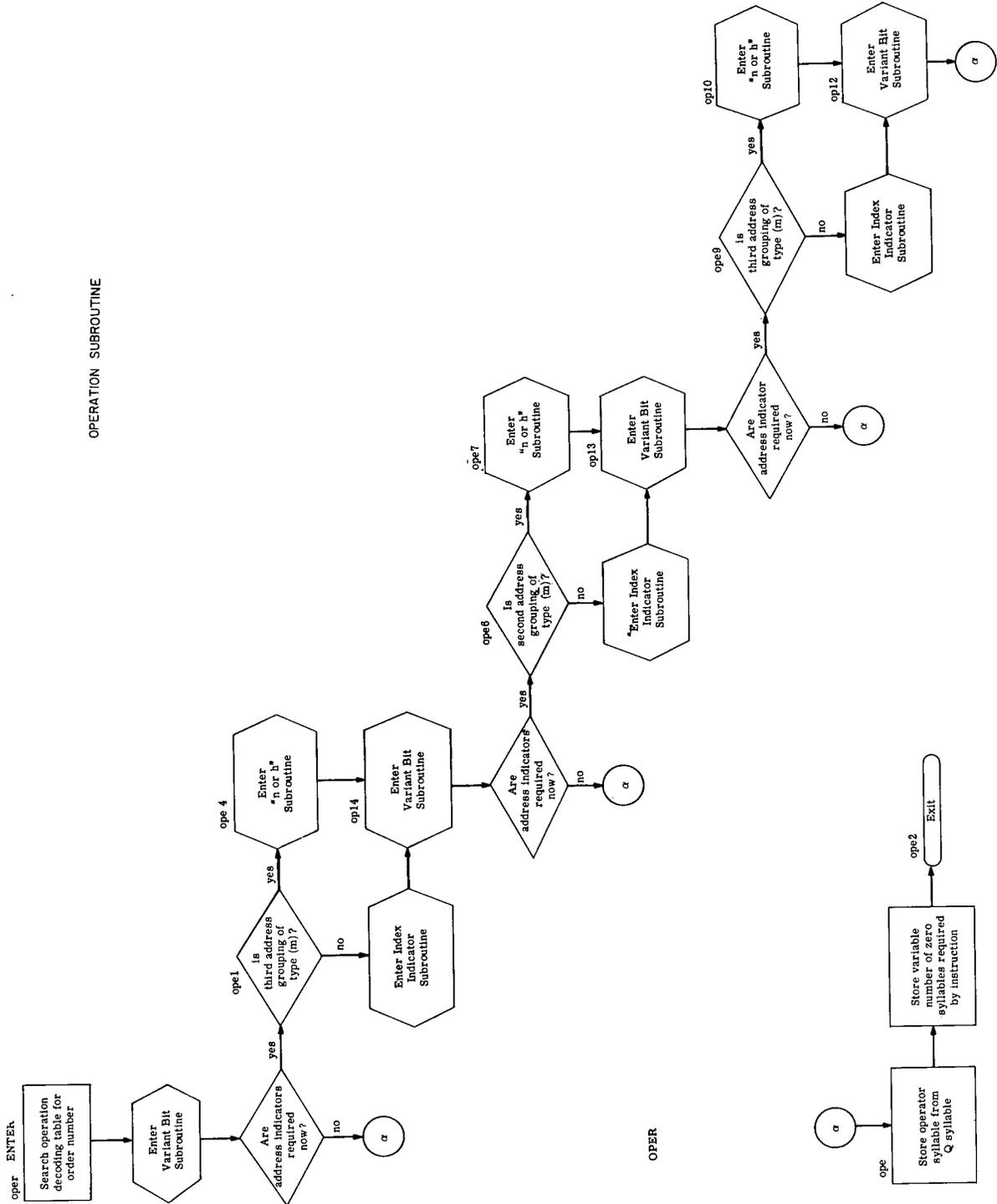
STORE Q SUBROUTINE



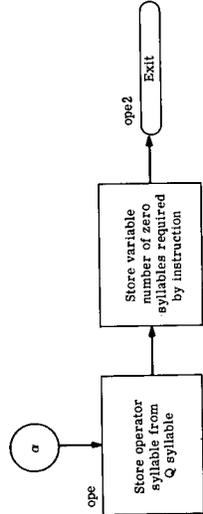
STORE NOP SUBROUTINE



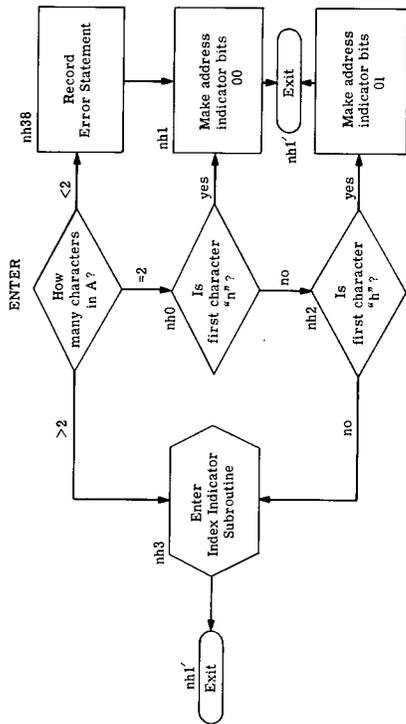
OPERATION SUBROUTINE



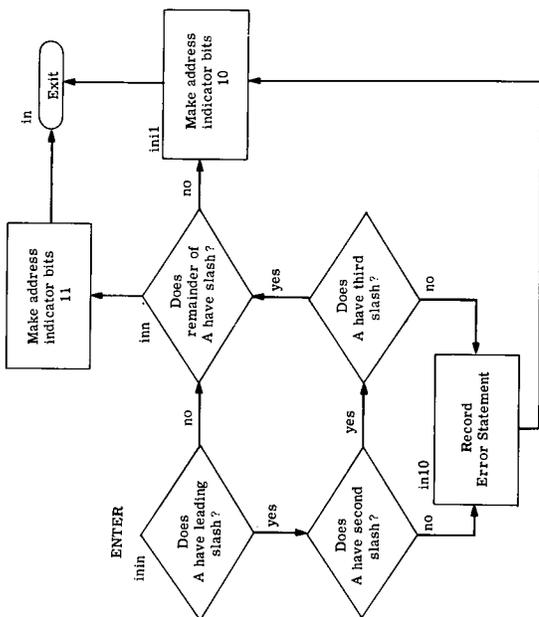
OPER



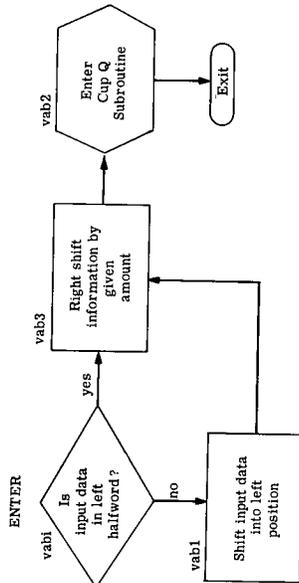
"N OR H" SUBROUTINE



INDEX INDICATOR SUBROUTINE

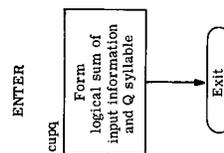


VARIANT BIT SUBROUTINE

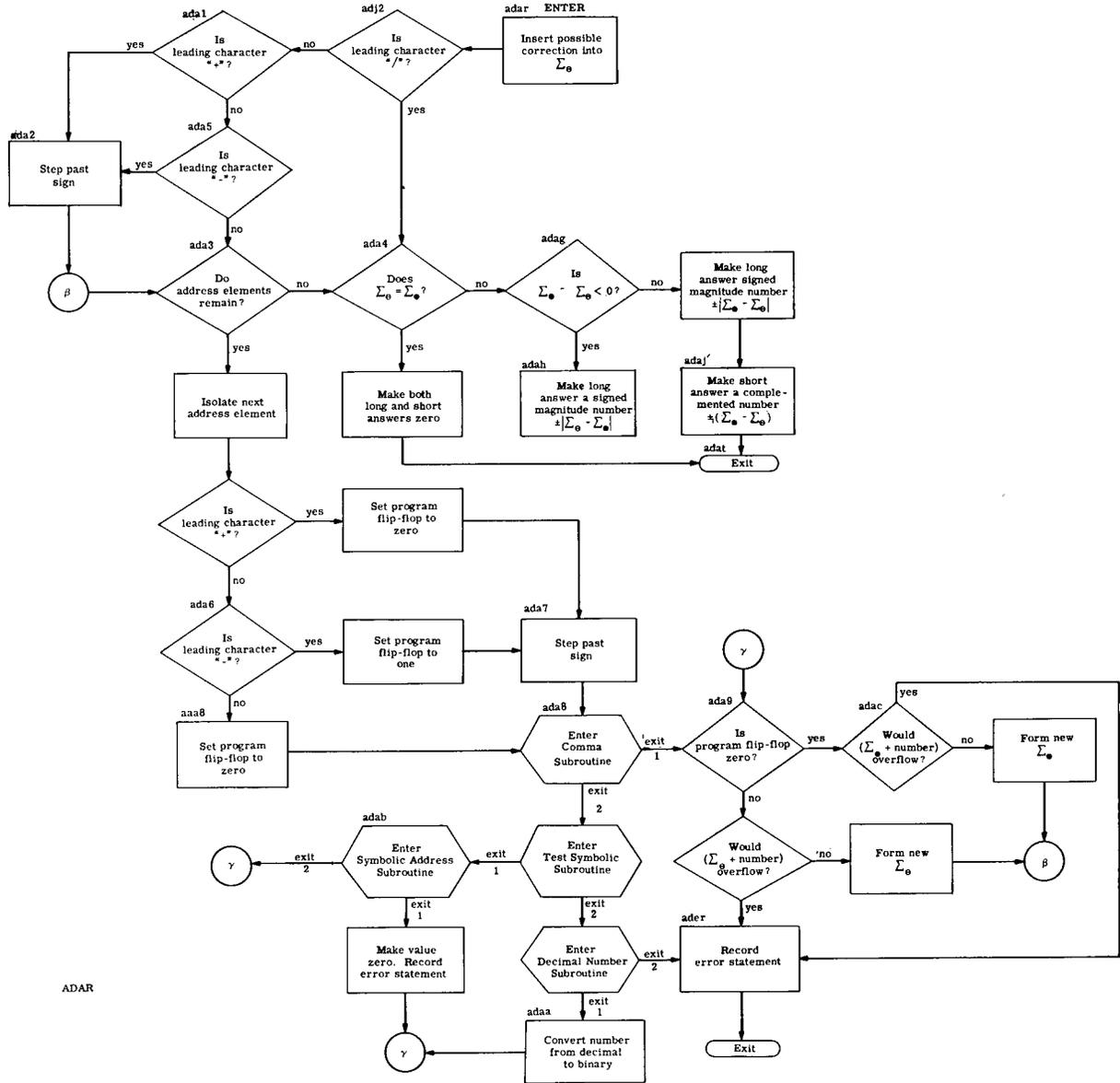


VABI

CUP Q SUBROUTINE

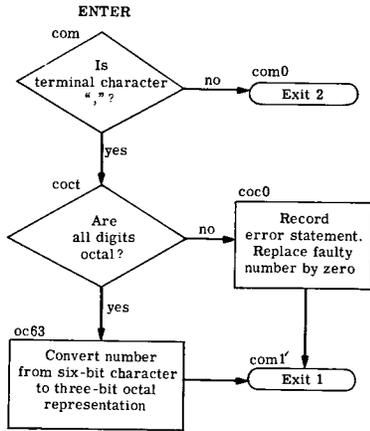


ADDRESS ARITHMETIC SUBROUTINE

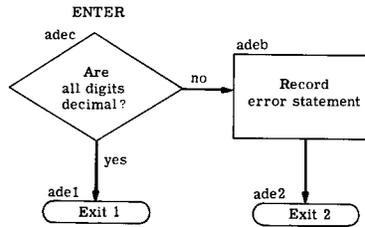


ADAR

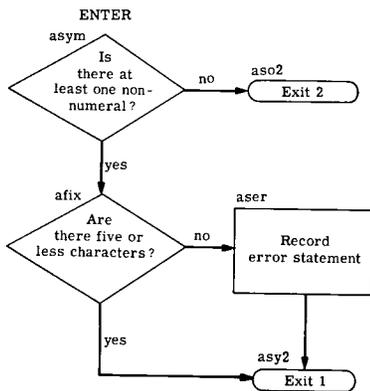
COMMA SUBROUTINE



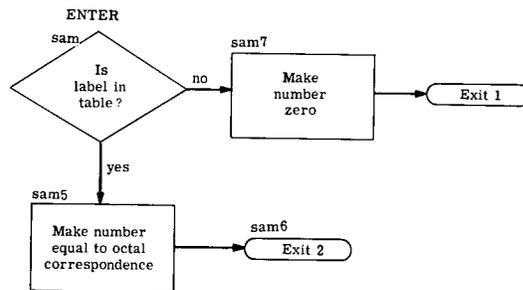
DECIMAL NUMBER SUBROUTINE



TEST SYMBOLIC SUBROUTINE



SYMBOLIC ADDRESS SUBROUTINE



## Appendix C

### SUMMARY OF NABUR RULES

#### I. NABUR WORDS

A NABUR word contains one and only one NABUR instruction, which may be regular, quasi-regular, or pseudo-, depending upon the requirements of the source program. It consists of a three-letter operation symbol and three address groupings, the last one, two, or three of which are to be omitted if not required to specify the operation.

#### II. LABELS

A. A "label" is a string of five or less strictly alphanumeric characters containing some letter other than "l," *e.g.*, "b2b5," "d," "alpha," "w1," "111t1," etc.

B. Operator syllables of instructions to be executed just after jumps must occupy the left-most syllable position of a computer word. This can be brought about by assigning a symbolic address to such NABUR words.

#### III. ADDRESSES

A. Contents of address syllables are determined from address quantities by the action of the Adar subroutine. NABUR subtracts the appropriate base register (BAR, BPR, or SAR) settings (found in "inbar," "inbpr," or "insar," respectively, as a result of "defines" by the programmer) from the M-, B-, and Ja-syllables respectively.

B. The automatic subtraction of the appropriate base register setting can be compensated for by adding either inbar, inbpr, or insar to the quantity in question, for example "Q1+inbar."

#### IV. SYMBOLS

A. The slash, "/", must precede all index quantities, which when utilized must follow all other address quantities within the grouping. For example, Q1/Q2/Q3/Q4, where Q1 is a memory address quantity, and Q2, Q3, and Q4 are quantities designating index registers.

B. The comma ",", coming at the end of a string of figures all less than or equal to 7, denotes an octal integer.

C. A space must separate address groupings; no space or tab is permitted within an address grouping.

D. A space must follow an operation symbol.

#### V. THIN-FILM DESIGNATIONS

A. The letter "n" means "normally step operand stack."

B. The letter "h" means "hold operand stack."

#### VI. THIN-FILM REGISTERS

A. Thin-film registers may be denoted by the octal codes given to them in the maps on pages C-2 and C-3 of Ref. 2. They also may be denoted by the decimal equivalents of these octal codes, by their common names as stated in the maps, or by new labels defined by the programmer. Limit registers require a reduction of their codes, modulo sixteen, in writing the Iv-syllable. This must be done beforehand by the programmer.

B. Regular instructions STF and LTF are used for single thin-film registers. If a group is required, quasi-regular instructions STW and LTW are used, and care must be taken to start the group with a key code.

#### VII. INDIRECT ADDRESSING

A. Indirect addressing is accomplished within memory syllables by prefixing a "+," sometimes understood, or a "-" to the address grouping.

B. Indirect addressing is accomplished within second or higher level, 16-bit absolute addresses in some computer word, say "spot," in the program, by using the INT pseudo-instruction with the reserved label "indi," as suggested in this example: "spot INT 65535+indi."

#### VIII. SNAG BIT

The snag bit of a second or higher level, 16-bit absolute address in some computer word,

say "spot," in the program, is set by using the INT pseudo-instruction with the reserved label "snag" as suggested in this example:

"spot INT 65545+indi+snag"

**IX. QUASI-REGULAR INSTRUCTIONS**

**A. Thin Film:**

- STW "store thin word"
- LTW "load thin word"

**B. Shifts**

- DDA "drop-off, double, arithmetic"
- DDL "drop-off, double, logical"
- DSA "drop-off, single, arithmetic"
- DSL "drop-off, single, logical"
- EDA "end-around, double, arithmetic"
- EDL "end-around, double, logical"
- ESA "end-around, single, arithmetic"
- ESL "end-around, single, logical"

**C. Index Limit Compare:**

- XNN "none"
- XEQ "equal"
- XGR "greater than"
- XGE "greater than or equal"
- XLS "less than"
- XLE "less than or equal"

- XUE "unequal"
- XUC "unconditional"

**X. TRANSMIT MODIFIERS**

The reserved labels "pu, mu, xu, pr, mr, and xr" are interpreted with the following key:

- p "transmit plus"
- m "transmit minus"
- x "transmit with changed sign"
- r "transmit rounded"
- u "transmit unrounded"

**XI. SOURCE PROGRAM MESSAGES**

Type Message	Identification	Typed on Source Tape?	Reappears in Checking Output?
REMARK	"r" in remark indication field	yes	yes
COMMENT	New line beginning with "CR tab"	yes	no
ASIDE	"Blank" in remark indication field	no	no