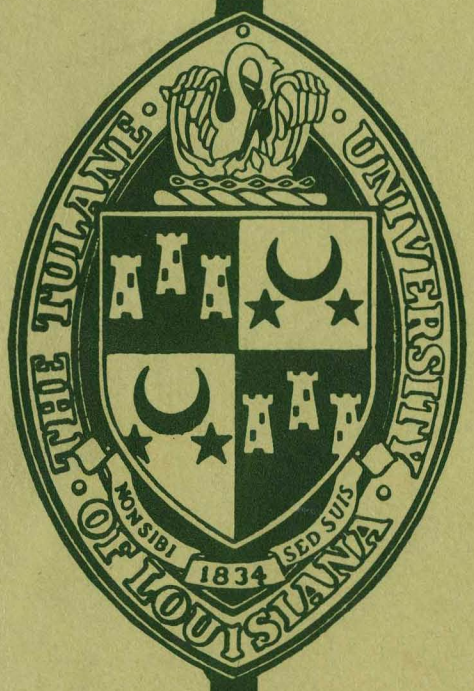
MINUTEMAN COMPUTER USERS GROUP



D17B COMPUTER DOCUMENTATION

MINUTEMAN COMPUTER

USERS GROUP

D17B COMPUTER

PROGRAMMING MANUAL

SUPPLEMENT

Edited By

Charles H. Beck

Professor and Director

SYSTEMS LABORATORY

Department of Electrical Engineering

School of Engineering

TULANE UNIVERSITY

REPORT MCUG-2-72

* D17B Operational Procedures
* D17B Programming Techniques
* D17B Programming Modules
* D17B Subroutines
* Corrections to the D17B Programming Manual

December 1972

PREFACE

This publication of the *MINUTEMAN COMPUTER USERS GROUP* serves as a supplement to the *D17B COMPUTER PROGRAMMING MANUAL*, Report MCUG-4-71. It contains many of the significant software developments completed in the Systems Laboratory at Tulane University. This supplement includes D17B operational procedures, programming techniques, the programming module set, and a description of some of the available D17B subroutines. A few additions and corrections to the *D17B COMPUTER PROGRAMMING MANUAL* are also included.

Arrangement of the material is specified by the table of contents on the following page which lists the first page number for each section. Lists of tables and figures are also included.

Additions and corrections will be distributed to MCUG members for the purpose of updating this publication as they are received. Please send correspondence to the Chairman of the MCUG at the following address.

Dr. Charles H. Beck
Professor of Electrical Engineering
Tulane University
New Orleans, Louisiana 70118

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

D17B OPERATIONAL PROCEDURES

1.1 Instructions for the Operator Control Panel

Light Display:

ooo ooo ooo ooo
ooo ooo ooo ooo

o o o
$S_4 S_2 S_1$

Each set of three adjacent lights represents a binary-coded octal digit. The upper row represents the left-half data/instruction word, and the lower row represents the right-half of the 8-digit word. The octal value corresponding to each digit equals

$$(S_4 \times 4) + (S_2 \times 2) + (S_1 \times 1)$$

Each S_n is 1 or 0 depending on the state of the corresponding light (ON or OFF). For example, if S_4 and S_1 are ON and S_2 is OFF, then the octal equivalent will be 4 + 1 = 5.

RUN-HALT:

RUN starts the computation at the location specified by a transfer instruction (TRA) in the I-register following either a MASTER RESET or COMPUTE code.
HALT places the computer in the Halt mode.

SINGLE CYCLE:
Precede by MASTER RESET or COMPUTE code

This will cause a single instruction to be executed whenever it is depressed. RUN-HALT must be in the HALT position.

MASTER RESET:

Set RUN-HALT to HALT

MASTER RESET initializes the following sequence:

(a) Establishes the $R_c V_c' K' J'$ mode.
(b) Places the computer in the Prepare to Operate submode.
(c) Achieves bit counter synchronization.
(d) Places a TRA to channel 00, sector 000 in the I-register.
(e) Resets the Phase Register.
(f) Clears the discrete output matrix.

Input Keyboard:

Keys are numbered 0 through 7 for the octal data/instruction inputs. When a key is depressed, the respective information is loaded into the least significant digit of the L-register, and the previous contents will be shifted one digit to the left.

FILL:

FILL produces a code which prepares the computer for the input of information in the Load mode.

LOCATION: This code causes the contents in the L-register to be transferred to the I-register which then operates as a location counter. The low order four octal digits represent the address.

ENTER: This code causes the contents in the L-register to enter the D17B memory location specified by the location counter in the I-register.

VERIFY: This code causes the computer to compare the information following the verify code with the information previously stored in corresponding locations in the computer memory. The Sb2 light will be ON whenever a disagreement is detected.

COMPUTE: This code places the computer in the Prepare to Compute submode. Computation will start at the location specified by a TRA instruction in the I-register when the RUN-HALT switch is placed in RUN.

CLEAR: This code causes the contents of the L-register to be cleared.

Memory Display Select: Any of the one-word memory loops within the D17B Computer may be monitored on a 24-bit binary display or an 8-digit octal numeric light display. The desired loop may be monitored by setting the memory display switch to the appropriate setting for the A-, L-, I-, U-, or N-loop. This switching can be accomplished on certain control panels by depressing the appropriate lighted push-button. An additional push-button is provided to permit the operator to monitor and display the contents of any word stored in the entire memory of the D17B. If this push-button is depressed, then the contents of any memory location specified by four octal coded lever switches can be monitored on the 24-bit binary display or an 8-digit octal numeric light display.

These displays will be particularly useful for debugging or trouble-shooting computer programs. They will also be useful for determining proper operation of the D17B.

1.2 Sequence of Operations for the Load Mode

The following procedure must be performed to load program and data characters into the D17B Computer.

1. Apply power to the computer, cooling system, I/O devices, and the various interface units.

2. Set RUN-HALT to HALT.

3. Set WRITE to ENABLE if the write heads for channels 00 through 46 are to be enabled. Set DISCRETE to ENABLE if the write head for channel 50 is to be enabled. In most cases these will always be set to ENABLE, or they may be hard-wired to ENABLE.

4. Push MASTER RESET and release it. The I-register (or loop) should read 50000000.

5. Enter a FILL code.

6. A sequentially addressed block of information representing data and/or instructions may be entered into memory as follows:

   (a) Enter the 4-digit octal location which represents the first address of the block of information. This will be loaded into the right-most digits of the L-register. A CLEAR code can be used to clear the L-register, but this is not necessary. The character presently being loaded will appear as the right-most digit of the L-register. The previous contents will be shifted one digit to the left with the left-most digit being shifted out as an overflow.

   (b) Enter a LOCATION code. The first address location will now appear in the right-most half of the I-register.

   (c) Enter the 8-digit octal data or instruction word into the L-register.

   (d) An ENTER code will cause the contents of the L-register to pass through the A-register and be stored in the memory address location specified by the I-register contents. The address in the I-register will be incremented by one.

   (e) For each additional sequential location, repeat steps (c) and (d). Repeated execution of step (d) will continue to increment the address.

7. If the next location is not sequential, i.e., a new block of information is started, repeat the procedure listed in step 6.

8. To verify that the information has been properly entered into memory, a VERIFY code should be entered followed by a reloading of the program and data as in step 6. When the ENTER code is processed, Sb2 will set (which results in turning on the indicator light) if the contents of memory are different from the corresponding information being loaded.

A flow chart for this sequence of operations is shown in Figure 1-1.

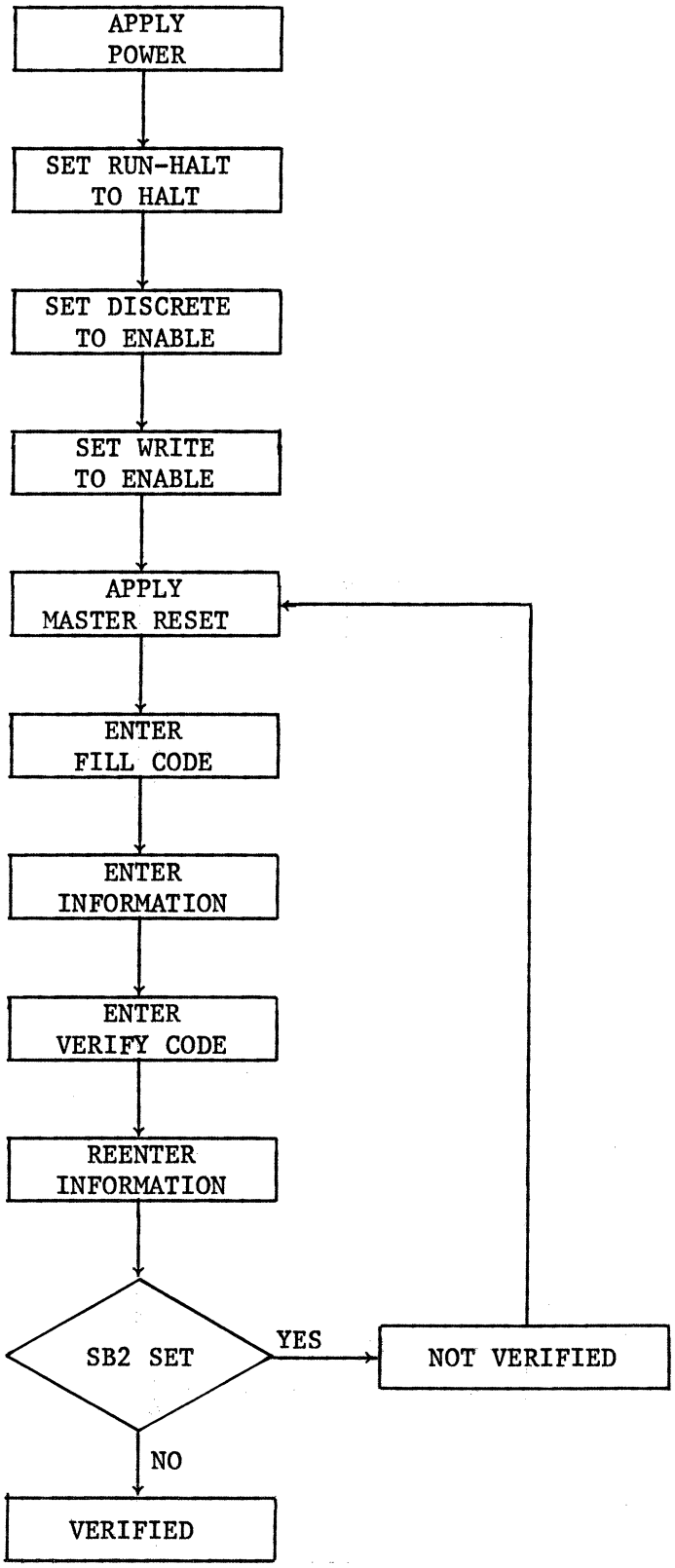


Fig. 1-1. Sequence of operations for the load mode.

1.3 Sequence of Operations for the Compute Mode

The following procedure must be performed to execute a program on the D17B Computer.

1. Repeat steps 1 through 4 of the Load Mode procedure.

2. Place RUN-HALT in RUN. At this point the computer will automatically transfer to channel 00 sector 000 and begin execution. This process takes place because 50000000 was initially in the I-register after the MASTER RESET operation.

3. To manually start a program in another location than channel 00 sector 000, place RUN-HALT in HALT. Enter eight octal characters corresponding to a TRA, transfer instruction, to the location where execution is to begin. Enter the LOCATION code and the COMPUTE code.

4. When RUN-HALT is placed in RUN, execution will begin at the desired memory location.

5. For single cycle operation perform the procedure listed in step 1 or 3 depending on the location where execution is to begin. If the starting location is channel 00 sector 000, then proceed to step 1. Otherwise, start with step 3. One instruction will be executed each time SINGLE CYCLE is depressed.

If it is desired to automatically execute a program which starts in a location other than channel 00 sector 000, then a TRA instruction to the desired channel and sector can be loaded in channel 00 sector 000. A flow chart for this sequence of operations is shown in Figure 1-2.

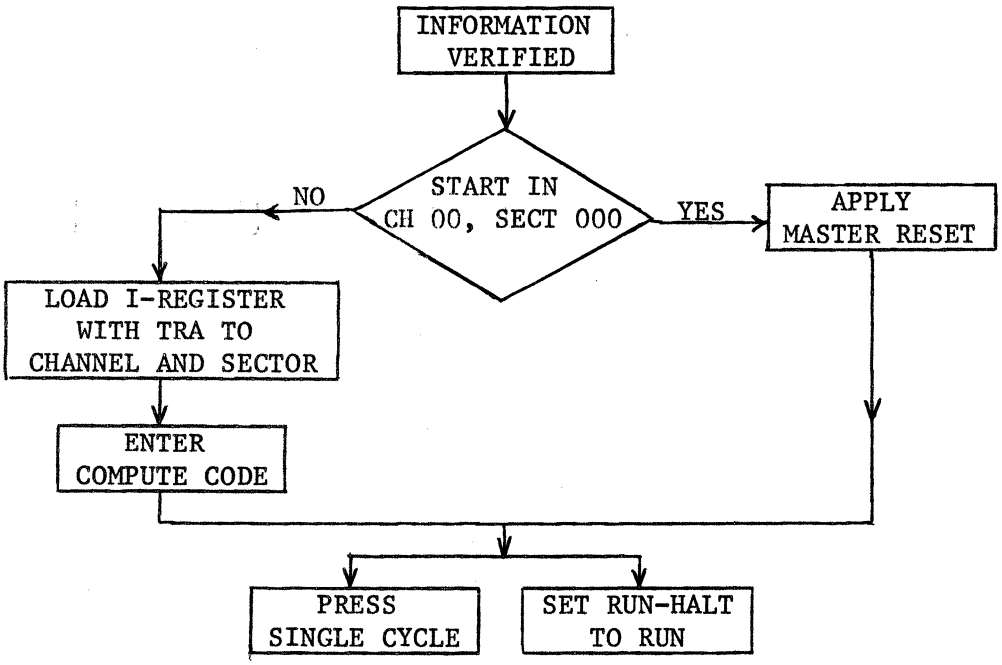


Fig. 1-2. Sequence of operations for the compute mode.

1.4 Minuteman D17B Computer Checkout Procedure

Before turning on the computer, check to see that all cables appear to be connected properly. After the computer has been inspected, begin the following checkout procedure. The left column gives the procedure to be followed, and the right column shows the correct status of the system. If the system does not show the same status as given here, turn off the computer and repeat the steps listed here again. Wait for at least two minutes before turning on the computer again. If the system still operates incorrectly, the trouble-shooting procedures should be followed.

PROCEDURE	SYSTEM STATUS
1. Place the ON-OFF switches for all power supplies and I/O equipment to OFF.	
2. Provide the necessary power line voltages.	
3. Place RUN-HALT in HALT.	
4. Place the 28 V Power Supply ON-OFF switch in the ON position. Turn on all I/O equipment and the related interface units.	The memory motor should rotate. The ammeter on the 28 V Power Supply should indicate approximately 20 A initially. It will then drop to approximately 18 A.
5. Press MASTER RESET and FILL.	I-register shows 50000000.
6. Press CLEAR.	L-register shows 00000000.
7. Press LOCATION.	I-register shows 00000000.
8. Press ENTER.	A-register shows 00000000. I-register advances one count, i.e., it shows 00000001.
9. Press in sequence the keys for 0, 1, 2, 3, 4, 5, 6 and 7.	L-register shows 01234567.

If the system status does not correspond to that given in steps 5 thru 9, repeat the sequence beginning with step 5 before resorting to powering up the system again.

CHAPTER 2

D17B PROGRAMMING TECHNIQUES

2.1 Minimal Delay Coding (MDC)

The location of successive instructions and their operands is extremely important because these locations are arranged serially within the various channels on the disc. With MDC an instruction can be completed in the number of word times equal to the basic execution time of that instruction. Therefore, to execute a number of sequential instructions in the minimum number of word times, the instructions should be separated by n-1 sectors where n is the basic execution time of the first instruction of each pair of these instructions. For MDC, it is always preferrable to have the operand available on the next sector following the location of this first instruction.

A few pertinent comments concerning MDC are listed in the following:

a) It will be desirable to utilize flag storing and the rapid-access loops as much as possible. These loops are the U-loop for one word (Unit), the F-loop for four words (Four), the E-loop for eight words (Eight), and the H-loop for sixteen words (Hexadecimal). The V-loop and the R-loop are two additional four-word rapid-access loops. While it is not possible to flag store to these loops, the standard STO instruction may be used. Data may be stored in either full or split-word format in the V-loop. However, data may only be stored in the split word format in the R-loop. Also, the left or most significant half of R.0 is not available for data storage. It will usually be required to reserve the L-loop for temporary data storage, for use with the ANA instruction, and for multiply operations.

b) When Operand Address Modification is involved, it will be especially helpful to use the loops for the operand address. In order to equalize the execution time and obtain MDC in the overall sense, the "Worst Case" (WC)

consideration can serve as a guideline. The WC consideration is to assure MDC for the worst possible operand location. Therefore, there will be no delay equal to a disc revolution in any case. Caution must be observed concerning the distance between the location of instructions with modified addresses and the locations of the next instructions to be executed. It is always appropriate to make the distance between these instructions at least equal to the word-length of the specific loop used if the full loop is to be involved. If only a portion of a loop is involved, a similar WC consideration can also be applied. The example shown in Table 2-1 illustrates this technique in detail.

SECTOR LOCATION	OP	C,S S_p
L + 0	CLA	L + 1, +2
L + 1	"Data"	
L + 2	STO	E.0, +12
L + 12	CLA	L + 2, +13
L + 13	ADD	L + 14, +15
L + 14	00000001	
L + 15	STO	L + 4, A

A: Next instruction after the Module.

Table 2-1. Example of operand address modification.

In the Data Storage Module shown in Table 2-1, the STO instruction at L + 2 is the modified address instruction. For example, if the full E-loop is involved, the next CLA instruction will not necessarily be executed during the current disc revolution for every sector in the E-loop unless the next instruction is located at L + 12. For L = 0 in the module illustrated in Table 2-1, the worst case of operand location is E.2. In this case operand agreement is found in sector 12, the sector of the CLA instruction. Therefore, equalization of execution time and the obtaining of MDC in the overall

sense have been preserved.

2.2 Optimum Storage Programming

Minicomputers such as the D17B have small memories. It will therefore be desirable to make full use of the available memory in a sophisticated D17B program. Utilization of the rapid-access loops is one of the most important quidelines for optimum storage programming. There are also other considerations such as the following:

a) Use of the same channel for instructions and data storage provides for efficient use of memory. The next sector can normally be used for data, and the following sector can be used for the location of the next instruction. Unless minimum execution time is of extreme concern, this programming technique is desirable because it will not waste any of the available memory. It also offers the advantage of keeping track of the data in the same channel as the instructions. In this manner, memory space can be saved; but longer execution time will be the trade-off.

b) It is desirable to avoid transferring to locations with sector addresses that are less than the present address unless a program loop is encountered. It is permissible to leave certain memory locations vacant to satisfy MDC. These locations can often be used for other portions of the program. This offers memory saving, equalization time, and often time saving as well.

2.3 Scaling of Constants

Data are expressed in the fractional binary form in the D17B. The fixed binary point is located between T_{24} and T_{23} for the full-word data format; the two fixed binary points for the split-word data format are located between T_{24} and T_{23} and between T_{11} and T_{10}. However, if it is necessary for the programmer to assume a location for the binary point, he must appropriately allow for it in programming and in interpreting results.

Scaling must be considered whenever a multiplication or a division is involved. If the binary point has been interpreted by the programmer as being located n-bits to the right of the original position, then he should shift the product left by n-bits on multiplication, and the quotient should be shifted n-bits to the right on division.

The binary point for a 24-bit data word can be assumed to be located in any position. If, for example, it is located between T_{12} and T_{13}, this will provide a total of 11-bits for the integer portion, 12-bits for the fractional portion, and one sign bit. The rules for floating point multiplication and division are applicable to the binary number system also. This scaling technique will be helpful in manipulating and interpreting data.

When a multiplication operation is executed using scaled data in the configuration specified previously, an 11-bit left-shift must be applied to the product. If a division operation is executed, an 11-bit right-shift must be applied to the quotient. The required shift can be partially or totally executed on the two operands as well as on the result of the multiplication or division operation as long as the sum of the bit-shifts equals the total number of required shifts. For instance, in the previous example, a 2-bit left-shift for the multiplicand, a 3-bit left-shift for the multiplier and a 6-bit left-shift for the product will yield a total of an 11-bit left-shift and produce the correct binary point interpretation for the true product. It is preferable to shift, at least partially, in advance to prevent data overflow or underflow.

It is the responsibility of the programmer to make sure that the magnitudes of the data values do not overflow or underflow throughout the various calculations. For example, when the Von Neumann division subroutine is used, caution must be exercised to assure that the absolute value of the divisor is greater than that of the dividend.

2.4 Coding Examples

Programming is commonly accomplished using the basic D17B machine language instructions. D17B instructions are three-address instructions; i.e., each instruction contains the location of the next instruction, the flag storage location, and the operand location. D17B programming is a somewhat unique experience. It may seem tedious and time-consuming at first by some. However, after becoming familiar with the instruction set and the programming techniques, the programmer will find D17B programming to be an enjoyable experience. Increased program execution speed and memory saving are two distinct advantages of machine language programming. The D17B instruction set and some preliminary programming techniques have been presented previously in the D17B Programming Manual, Report MCUG-4-71.

Three coding examples are shown in Table 2-2 for unflagged instructions, and three more are shown in Table 2-3 for flagged instructions. Examples 1, 3, and 4 illustrate MDC.

2.5 Flag Store Codes

The D17B has a "flag store" mode which provides for simultaneously storing the previous contents of the Accumulator in certain specified channels coincident with the execution of an instruction. This feature eliminates the need for an additional instruction to perform this frequent operation. Some comments concerning these flag store codes are listed in Table 2-4.

2.6 Exceptions for Instruction Codes

The complete instruction set has been illustrated in the D17B Programming Manual, Report MCUG-4-71. There are certain exceptions for specific instructions which, if ignored, may lead to faulty results or inappropriate computer operation. These exceptions are summarized in Table 2-5.

Example 1

LOCATION	OP	(C,s) OPERAND; S_p
02,023	CLA	04,024;024

OP Code = 1001 (44)
Flag = 0
S_p = 0 010 100 (024)
C,s = 000 100 010 100 (0424)

Binary Instruction = 100 100 010 100 000 100 010 100

Octal Instruction = 4 4 2 4 0 4 2 4

Direct Method = 44240424

Example 2

42,131	SUB	10,170;132

OP Code = 74
Flag = 0
S_p = 132
C,s = 1170

Direct Method = 75321170

Example 3

20,070	ALS	22,006;077

OP Code = 00
Flag = 0
S_p = 077
C,s = 2206

Direct Method = 00772206

Table 2-2. Coding examples for unflagged instructions.

Example 4

LOCATION	OP	(C,s) OPERAND; S_p
02,001	CLA*L	04,002;002

OP Code = 1001 (44)
Flag = 1
S_f = 101
S_p = 0010
C,s = 000 100 000 010 (0402)

Binary Instruction = 100 111 010 010 000 100 000 010
Octal Instruction = 4 7 2 2 0 4 0 2
Direct Method = 47220402

Example 5

16,143	ADD*F.2	04,166;175

OP Code = 1101 (64)
Flag = 1
S_f = 001
S_p = 1101
C,s = 000 101 110 110 (0566)

Binary Instruction = 110 110 011 101 000 101 110 110
Octal Instruction = 6 6 3 5 0 5 6 6
Direct Method = 66350566

Note: This instruction will flag store the contents of the Accumulator to F.2 because the last two bits of the operand are 10. The next instruction will come from 1775 because the first 1101 combination after Sector 166 is Sector 175.

Example 6

20,000	SUB*E.1	40,001;035

Note: This instruction cannot be coded in D17B machine languate. The operand is correct for flag storing to E.1, but the next instruction is located more than 16 sectors following the location of the operand.

Table 2-3. Coding examples for flagged instructions.

S_f	CHANNEL	COMMENTS
00	No Flag Operation	
02	F-Loop (52)	4 words, location of data storage determined by binary coding of last two bits of the operand address. 00, 01, 10 and 11 correspond to words F.0, F.1, F.2 and F.3, respectively.
04	Telemetry Output	Under program control a timing signal is issued which signifies that there is information in the Accumulator which is to be read by the external device which receives the timing signal.
06	Modifiable Channel (50)	128 words, addressable when DDC is set true. Location of data storage determined by the sector address of the operand minus two.
10	E-Loop (56)	8 words with intermediate read head. Location of data storage determined by binary coding of last three bits of the operand address. Words are available every four word times; i.e., only agreement of last two bits is required.
12	L-Register (64)	Available at all times except for MPY, SMP, STO, TRA, TMI or ANA instructions.
14	H-Loop (54)	16 words with intermediate read head. Location of data storage determined by binary coding of last four bits of the operand address.Words are available every eight word times; i.e., only agreement of last three bits are required.
16	U-Loop (60)	1 word, available at all times except for STO instruction.

Table 2-4. Flag store codes.

INSTRUCTION	EXCEPTIONS
MPY, SMP	The L-Register is not available for a flag instruction.
ANA	The L-Register is not available for a flag instruction
COA	This instruction is not defined when s is zero.
TRA	Channel address cannot be 64(L), 70(V), or 72(R).
TMI	Channel address cannot be 64(L), 70(V), or 72(R). When A > 0, all flag codes are defined only if MDC is used for TMI.
STO	A-Register, L-Register, and U-Loop are not available. Channels 00 through 46 are not available for EWC', and Channel 50 is not available for DDC Channel 50, the F-Loop, the E-Loop, and the H-Loop are not defined as operands for flagged instructions except for S_f = 12 or 16.
HPR	MDC must be used.

Table 2-5. Exceptions for instruction codes.

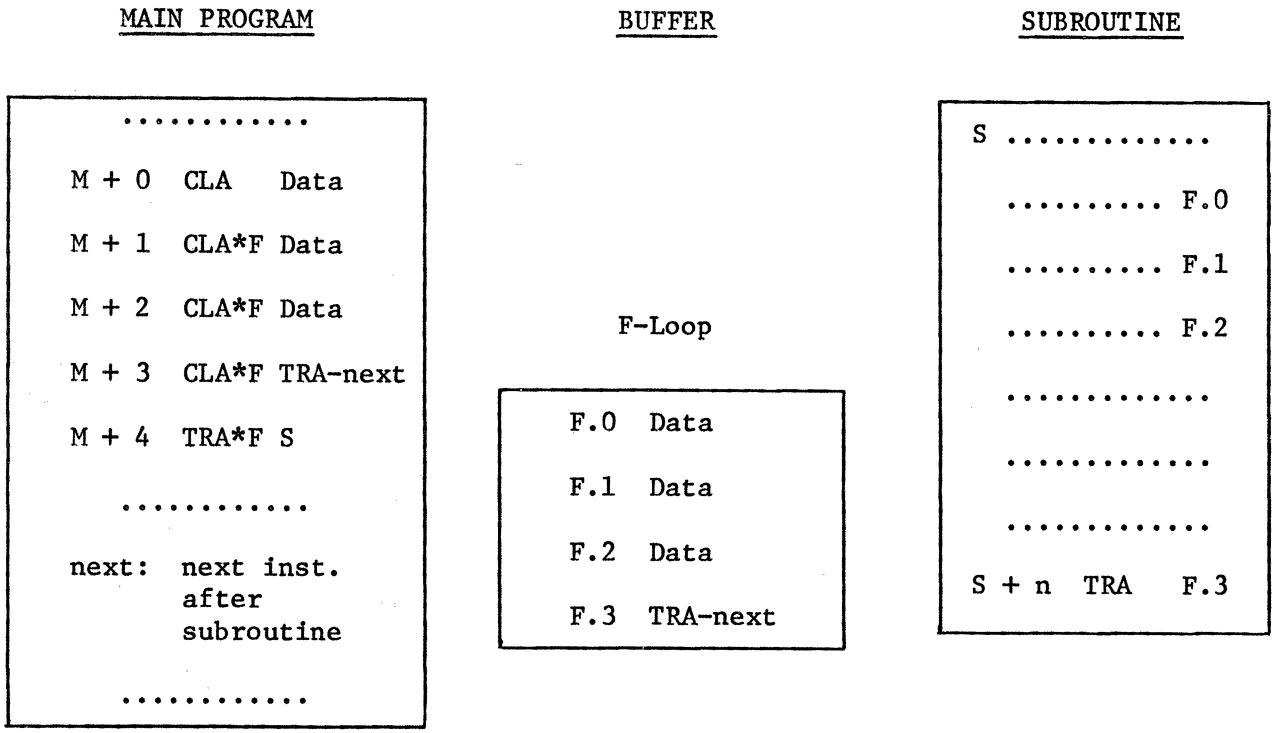
2.7 Subroutine Linkage

In general, any program executed on a digital computer is composed of several previously written subroutines called in a proper sequence by a main program. Because the D17B does have limited memory capacity as is the case with most minicomputers, it may be both necessary and convenient to use the same subroutinerepeatedly, with different data values, in several steps during a particular program. Since it is unnecessary to duplicate the list of instructions in a subroutine each time it is used, a multiple subroutine linkage (MSL) must be developed so that the computer may execute the subroutine one or more times during a particular program with one or more different data values.

In many programs executed on the D17B, one subroutine may be used at several different times during a particular program. The program execution time may be increased slightly if MSL is used, but this linkage technique is always desirable unless the execution time is a critical consideration. However, if the subroutine contains a small number of instructions, it then becomes questionable as to whether it is worthwhile to program a linkage and then transfer to the subroutine or to just repeat the instructions.

A MSL basically needs to perform three operations. First, it must give the subroutine any needed data values. Second, it must tell the subroutine where to return in the main program after it has completed its calculations. Finally, the linkage must transfer execution to the beginning of the subroutine. The flag store capability of the D17B will aid in developing a concise linkage. The 4-word F-loop will be used as a buffer between the main program and the subroutine. Table 2-6 shows a block diagram of the proposed linkage.

When the programmer wants to call a certain subroutine, he will first load any input data required by the subroutine, defined as external data by



Note: Last two bits of S must be 11.

Table 2-6. Standardized D17B subroutine linkage.

TUSL, into words 0, 1, and 2 of the F-loop via the flagged CLA instructions. This means that up to three input data words may be transferred from the main program to the subroutine. However, recall that the subroutine probably has access to many other data words which are fixed values (i.e. internal data which do not vary with input data) used for calculations internal to the subroutine. It would also be possible to use a larger loop, say the E-loop or the H-loop, except that loading the data into the specific word within the loop might mean loss of memory space due to the necessary coding. Therefore, in some cases, the E- or H-loops might not be more economical, in terms of memory space, than the normal CLA-STO, CLA-STO, ... process.

After the data have been stored in words 0, 1, and 2 of the F-loop (represented as F.0, F.1, F.2), a TRA instruction is loaded into F.3. The operand of this instruction will be the location of the next instruction after the subroutine has been completed. A TRA instruction to the beginning of the subroutine concludes the call sequence. It is important to note that this TRA instruction may actually be used to flag store the accumulator. As shown in Table 2-6, the TRA instruction in location M+4 flag stores the TRA-next to word F.3. This means that the linkage is five instructions compared to nine for the CLA-STO, CLA-STO, ... method.

When the subroutine requires certain external data from the main program, it will simply use F.n as the operand. That is, the data locations will be fixed when the subroutine is written, and the program will load the values into the proper location in the F-loop during the call sequence. Because the F-loop is used for the data, the longest delay, if MDC is not used for coding the operand, will be four word-times. At the end of every subroutine, the program execution will transfer via a TRA instruction to F.3. Location F.3 will contain another TRA instruction, loaded during call, which will transfer

execution back to the proper location in the main program. If used consistantly, MSL should prove to be efficient.

For the non-multiple modular cases, i.e., none of the various subroutines will be utilized repeatedly, linkage simply means to chain appropriate modules as specified by the program flow chart. Typical modules are given in Chapter 3. Programwise, this can be achieved by designating the next instruction location for the last instruction of the previous subroutine or module to be the starting location of the later subroutine or module. The complete Non-Multiple Modular Linkage (NML) can be assembled in the same manner.

2.8 Address Modification

The disc memory of the D17B is used for the storage of both data words and instructions. It is this feature which makes address modification possible. When a series of data values are operated on consecutively by a certain arithmetic operation, address modification becomes very desirable for the sake of program-simplification. In performing address modification, the instructions are regarded as data words which can be modified as needed. Data storage and retrieval are typical examples of situations where address modification is needed. These operations are illustrated in Table 2-7.

OPERATION	LOCATION	OP	C,s; S_p	COMMENTS
Data Storage	L + 0	CLA	L'+ 1, +1	D_n' in L'+ 1
	L + 1	STO	N'+ 2, +2	D_n' stored in N'
	L + 2	CLA	L + 1, +3	
	L + 3	ADD	L'+ 4, +4	1 in L'+ 4
	L + 4	STO	L + 3, A	STO modified stored in L + 1
	(Data storage starts at N')			
Data Retrieval	L + 0	CLA	L + 3, +1	
	L + 1	ADD	L'+ 2, +2	1 in L'+ 2
	L + 2	STO	L + 5, +3	CLA mod. in L + 3
	L + 3	CLA	L'+ 3, A	Retrieval starts at L'+ 4

Table 2-7. Typical address modification examples.

In the examples shown in Table 2-7, MDC has only been preserved for certain data values. This may be undesirable for those cases where timing is a critical consideration. Address modification can be accomplished in these cases by using the rapid-access loops. The examples in Table 2-8 show how data storage or data retrieval can be achieved without excessive delays of several disc revolutions.

OPERATION	LOCATION	OP	C,s; S_p	COMMENTS
Data Storage	0000	CLA	0001, 02	
	0001	STO	E.2, 01	
	0002	CLA*F	0003, 04	"STO inst." in F.3
	0003	TRA	0006	"TRA inst." in F.1
	0004	CLA*F	0205, 05	"D(n)" in 0205
	0005	TRA	F.3	
	0006	CLA	F.3, 07	
	0007	ADD	0210, 12	1 in 0210
	0012	CLA*F	0213, A	A: Next instruction

Note: 0000 through 0004 only for initialization.

Data Retrieval	0000	CLA	0001, 02	
	0001	CLA	E.1, 01	
	0002	CLA*F	0003, 04	"CLA inst." in F.3
	0003	TRA	A	A: Next instruction
	0004	CLA*F	0205, 05	"D(n)" in 0205
	--	--	-- --	
	1002	CLA	F.3, 03	
	1003	ADD	0204, 06	1 in 0204
	1006	STO	F.3, 07	
	1007	TRA	F.3	

Note: Channel 00 only for initialization.
Data retrieval starts at E.2.

Table 2-8. Examples of address modification using rapid-access loops.

CHAPTER 3

D17B PROGRAMMING MODULES

There are a total of 39 instructions available for D17B programming. Although the instruction set is relatively small compared to that available with large-scale general-purpose computers, this instruction set is quite sufficient if the instructions are used appropriately. The concept of a "Module" has been created in the Tulane Systems Laboratory through experience gained in D17B programming. Here, modules, each of them a set of several instructions, represent different arithmetic calculations or logic operations which are useful in D17B programming. Each module may be equivalent to more than a single instruction used with a more sophisticated computer or a higher level programming language.

Programmers are always urged to draw a flow chart before writing any program. Furthermore, it is suggested that flow charts be prepared in such a manner that the module set can be used to realize each block of the flow chart. Considerable experience has shown that, in most cases, flow chart blocks can always be realized using a set of typical modules. Of course, the size of module block can be extended, in the limit, to the complete flow chart itself. The usefulness of the respective module will then be limited, however, because of the lack of repeatability.

The modules can usually be divided into three categories which are the Arithmetic Operations, Control Operations, and Input/Output Operations. The most useful modules within each category based on experience in the Tulane Systems Laboratory are listed in Table 3-1. The mnemonic symbol for each module is shown in parentheses. The explanations and instruction listings for the various modules are given starting in Section 3-1. Since the modules are supposed to be used repeatedly in D17B programming, variable locations,

which are designated with letters such as L, A, B, C, etc., are used in the standard format notation.

It is the responsibility of the programmer to assure that MDC is preserved. Throughout this Chapter, L and L' represent memory locations with identical sector numbers. Familiarity with the module set is just as significant as familiarity with the instruction set and is strongly recommended.

Arithmetic Operations

1. Constant Initialization (CIN)
2. Decreasing Counter (DEC)
3. Increasing Counter (INC)
4. Time Delay (TDE)
5. Mid-Point Interpolation (MPI)
6. Averaging (AVE)
7. Data Storage (DST)
8. Data Retrieval (DRE)
9. Normalized Gradient (NGR)
10. Absolute Value (ABS)
11. Masking (MAS)
12. One's Complement (ONC)
13. Logical OR Operation (LOR)
14. Data Rotation (DRO)
15. Packing (PAC)
16. Unpacking (UNP)

Control Operations

1. Cycle End Test (CET)
2. Bounds Test (BOT)
3. Rising Test (RIT)
4. Falling Test (FAT)
5. Transfer on Non-Zero (TNZ)
6. Transfer on Positive (TPO)
7. Identity Check (IDC)
8. Transfer on Negative Product (TNP)

Input/Output Operations

1. Character Output (CHO)
2. Error Identification (EID)
3. Digital-to-Analog Conversion (DAC)
4. Analog-to-Digital Conversion (ADC)
5. Pulse Waveform Generation (PWG)
6. Interactive Input (INI)

Table 3-1. D17B programming modules.

3.1 Arithmetic Operations

CIN Constant Initialization

This is used for initializing the constants at the beginning of a program.

SECTOR LOCATION	OP	C,S	S_p
L + 0	CLA	L'+ 1,	+1
L + 1	CLA*L	L'+ 2,	+2
L + 2	CLA*U	L'+ 3,	+3
L + 3	CLA*F	L'+ 4,	+4
L + 4	CLA*F	L'+ 5,	+5
L + 5	CLA*F	L'+ 6,	+6
L + 6	CLA*F	L'+ 7,	A

$$L + 7 \leq A \leq L + 26$$

DEC Decreasing Counter

The counter decreases 1 after the module is executed.

L + 0	CLA	L'+ 1, +1	Initial count in L'+ 1
L + 1	SUB	L'+ 2, +2	"1" in L'+ 2
L + 2	STO	L'+ 3, A	A: Next instruction

INC Increasing Counter

The counter increases 1 after the module is executed.

L + 0	CLA	L'+ 1, +1	Initial count in L'+ 1
L + 1	ADD	L'+ 2 +2	"1" in L'+ 2
L + 2	STO	L'+ 3, A	A: Next instruction

TDE Time Delay

Time is delayed for $10(T + \frac{3 + D}{128})$ ms where D is the sector distance between L + 2 and A.

L + 0	CLA	L'+ 1, +1	"T" in L'+ 1
L + 1	SUB	L'+ 2, +2	"1" in L'+ 2
L + 2	TMI	A, +1	A: Next instruction

MPI Mid-Point Interpolation

This will perform the calculation D(n+1) = (D(n) + D(n+2))/2

L + 0	CLA	L'+ 1, +1	"D(n)" in L'+ 1
L + 1	ADD	L'+ 2, +2	"D(n+2)" in L'+ 2
L + 2	ARS	3201, +4	
L + 4	STO	L'+ 5, A	"D(n+1)" in L'+ 3
			A: Next instruction

AVE Averaging

Depending on whether n is a Kth power of 2, different techniques are used in finding the average of a set of values.

```
    L + 0       CLA      L'+ 1, +1     "Sum" in L'+ 1
    L + 1       ARS      32(k8),k+2
L + k + 2       STO      L'+k+3,  A    "AVE" in L'+k+1
                                       A:  Next instruction
```

If $n \neq 2^k$, then the division subroutine should be used.

DST Data Storage

A series of data values can be stored in consecutive locations by using address modification.

```
    L + 0       CLA      L'+ 1, +1     "D(n)" in L'+ 1
    L + 1       STO      N'+ 2, +2     "D(n)" stored in N'
    L + 2       CLA      L + 1, +3
    L + 3       ADD      L'+ 4, +4     "1" in L'+ 4
    L + 4       STO      L + 3,  A     A:  Next instruction
```

DRE Data Retrieval

This is the same as DST except that data are retrieved instead of stored.

```
    L + 0       CLA      L + 3, +1
    L + 1       ADD      L'+ 2, +2     "1" in L'+ 2
    L + 2       STO      L + 5, +3     "CLA inst." stored
    L + 3       CLA      L'+ 3,  A          in L + 3
                                       A:  Next instruction
```

Data retrieval starts at L'+ 4.

NGA Normalized Gradient

The normalized gradient is the function increment.

```
    L + 0       CLA      L'+ 1, +1     "D(n+1)" in L'+ 1
    L + 1       SUB      L'+ 2, +2     "D(n)" in L'+ 2
    L + 2       STO      G + 2,  A     G:  Gradient storage
                                       A:  Next instruction
```

ABS Absolute Value

A = -A if A < 0, but it is unchanged otherwise.

```
    L + 0       CLA      L'+ 1, +1
    L + 1       MIM       4400, +2
    L + 2       COM       4600,  A     A:  Next instruction
```

MAS Masking

If certain bits of a word are to be preserved with the remainder of the bits being cleared, a masking technique can be used.

```
L + 0      CLA       L'+ 1, +1    "Mask" in L'+ 1
L + 1      CLA*L     L'+ 2, +2    "Data" in L'+ 2
L + 2      ANA        4200,  A    A:  Next instruction
```

ONC One's Complement

Negative numbers are expressed in the 2's complement in the D17B. If the 1's complement is desired, the following module is useful.

```
L + 0      CLA       L + 1, +2
L + 1      Data
L + 2      COM        4600, +3
L + 3      SUB       L + 4,  A    A:  Next instruction
L + 4    00000001
```

LOR Logical OR Operation

Two binary variables can be logically ORed as $A + B = \overline{(\overline{A} \cdot \overline{B})}$.

```
L + 0      CLA          A, +1     A:  Data No. 1
L + 1      COM       4600, +2
L + 2      SUB       L + 3, +4
L + 3    00000001
L + 4      CLA*L        B, +5
L + 5      COM       4600, +6
L + 6      SUB       L + 7, +8
L + 7    00000001
L + 8      ANA       4200, +9
L + 9      COM       4600, +10
L + 10     SUB       L+ 11,  C    C:  Next instruction
L + 11   00000001
```

DRO Data Rotation

The locations of two respective data can be rotated using this module.

```
L + 0      CLA          A, +1
L + 1      CLA*U        B, +2
L + 2      STO       A + 2, +3
L + 3      CLA       6000, +4
L + 4      STO       B + 2,  N    N:  Next instruction
```

PAC Packing

This module packs an 8-bit right-justified word (x) into bits 9-16 of the U-loop. Two other 8-bit words were previously located in bits 1-8 and 17-24, and zeros were initially located in bits 9-16 of (x) and the U-loop.

```
L + 0      CLA      L'+ 1, +1    (x) in L'+ 1
L + 1      ALS       2210, +12
L + 12     ADD       6000, +13
L + 13     STO      L'+14,  A    Packed word in L'+12
                                 A:  Next instruction
```

UNP Unpacking

This module unpacks the middle 8-bit word from a word containing three 8-bit words. The word to be unpacked is assumed to be located in the U-loop, and the unpacked, right-justified 8-bit word will be located in the Accumulator.

```
L + 0      CLA      L'+ 1, +1    M1 in L'+ 1
L + 1      CLA*L        U, +2
L + 2      ANA       4200, +3
L + 3      ARS       3210,  A    A:  Next instruction
                                 M1 = 00177400
```

3.2 Control Operations

CET Cycle End Test

This module will check for the end of a loop-cycle.

```
L + 0      CLA      L'+ 1, +1    Counter in L'+ 1
L + 1      SUB      L'+ 2, +2    "n" in L'+ 2
L + 2      TMI          A, +3    DEC or INC in A
L + 3      CLA      L'+ 4, +4    Counter recovery
L + 4      STO      L'+ 3,  A    A:  Next instruction
```

BOT Bounds Test

This module tests for the condition that a specific word is out of range with respect to upper and lower bounds.

```
L + 0      CLA      L'+ 1, +1
L + 1      SUB      L'+ 2, +2    "Max" in L'+ 2
L + 2      TMI      L + 3, +7
L + 3      CLA      L'+ 1, +4
L + 4      SUB      L'+ 5, +5    "Min" in L'+ 5
L + 5      TMI      L + 6,  A    A:  Next instruction
L + 6      CLA      L'+ 7,  E    E:  EID module
L + 7      CLA      L'+ 8,  E    Error codes in
                                    L'+ 7 and L'+ 8
```

RIT Rising Test

This module tests for a rising trend in a series of data.

```
L + 0      CLA      L'+ 1, +1    "D(n+1)" in L'+ 1
L + 1      SUB      L'+ 2, +2    "D(n)" in L'+ 2
L + 2      TMI          A,  B    A:  Next inst. if NO
                                 B:  Next inst. if YES
```

FAT Falling Test

This module tests for a falling trend in a series of data.

```
L + 0      CLA      L'+ 1, +1    "D(n+1)" in L'+ 1
L + 1      SUB      L'+ 2, +2    "D(n)" in L'+ 2
L + 2      TMI          A,  B    A:  Next inst. if YES
                                 B:  Next inst. if NO
```

TNZ Transfer on Non-Zero

The module causes a transfer to a specific location if the contents of the Accumulator are non-zero.

```
L + 0      CLA      L'+ 1, +1
L + 1      MIM       4400, +2
L + 2      TMI          A,  B    A:  Next inst. if
                                       non-zero
                                 B:  Next inst. if zero
```

TPO Transfer on Positive

The module causes a transfer to a specific location if A > 0.

```
L + 0      CLA      L'+ 1, +1
L + 1      COM       4600, +2
L + 2      TMI          A,  B    A:  Next inst. if
                                       A > 0
                                 B:  Next inst. if
                                       A ≤ 0
```

IDC Identity Check

This module tests for the equivalence between two words.

```
L + 0      CLA      L'+ 1, +1
L + 1      SUB      L'+ 2, +2    "N" in L'+ 2
L + 2      MIM       4400, +3
L + 3      TMI          A,  B    A:  Next inst. if NO
                                 B:  Next inst. if YES
```

TNP Transfer on Negative Product

The module causes a transfer to a specific location if the product of two data values is negative.

```
L + 0      CLA        L + 1, +2
L + 1     Data A
L + 2      CLA*L      L + 3, +4
L + 3    40000000
L + 4      ANA         4200, +5
L + 5      ADD        L + 6, +7
L + 6     Data B
L + 7      TMI            C,  D     C:  Next inst. if
                                           A•B < 0
                                     D:  Next inst. if
                                           A•B ≥ 0
```

3.3 Input/Output Operations

CHO Character Output

This module transfers the contents of a specific word into the voltage output register and initializes the teletype to print out the character with ASCII Code corresponding to bits 17-24 of the A-register.

```
L + 0      CLA       L'+ 1, +1
L + 1      VOA        3000, +2    A24-17 → V18-11
L + 2      DOA        2613, +3
L + 3      DOA        2600,  A    A:  Next instruction
```

EID Error Identification

An error code will be printed when an error is detected, and the buzzer will be sounded using this module.

```
L + 0      CLA       L'+ 1, +1    Error code in L'+ 1
L + 1      STO       L'+ 2, +2
L + 2      DOA        2610, +3    Buzzer is pulsed
L + 3      DOA        2600, +4
L + 4      HPR        2200, +5
L + 5      TRA         CHO,  A    A:  Next instruction
```

DAC Digital-to-Analog Conversion

The analog voltage output corresponding to certain digital information is made available.

```
L + 0      LPR        7020, +1    Set P3
L + 1      CLA       L'+ 2, +2
L + 2      VOA        3400,  A    A:  Next instruction
```

ADC Analog-to-Digital Conversion

This module pulses the A-D converter and inputs the digital data to the D17B.

```
L + 0      DOA       2620, +1    Pulse A-D converter
L + 1      DOA       2600, +2
L + 2      DIB       5000,  A    A:  Next instruction
```

PWG Pulse Waveform Generation

This module causes the data (x) to be counted up or down depending on whether (x) is negative or positive respectively. Then (x) will switch between 00000000 and 77777777 thus generating a pulse train with pulse width of 10 ms and a period of 20 ms.

```
L + 0      CLA       L'+ 1, +1
L + 1      BOA        1000, +1
```

INI Interactive Input

This module inputs an 8-bit character from the peripheral I/O device under program control. X8C - X1C are connected to the 8-bit character information lines. X9C is connected to the character ready line.

```
L + 0      CLA       L + 1, +2
L + 1    00000400                1 in bit 9
L + 2      DIA*L      5200, +3   X9C-X1C → Accumulator
L + 3      ANA*U      4200, +4   Mask A9
L + 4      SUB       L + 5, +6
L + 5    00000400                1 in bit 9
L + 6      TMI       L + 0, +7   Was X9C = 1?
L + 7      DOA        2616,+10   Reset Ready Line
L +10      DOA        2600,  A   A:  Next Instr.
                                 Character located in U
```

CHAPTER 4

D17B SUBROUTINES

Various subroutines which have been useful for D17B programming have been developed by the staff of the Tulane University Systems Laboratory (TUSL). Each subroutine can be considered as an independent program which will execute a specific operation. The standardized subroutine linkage technique is illustrated in Section 2.4. As long as the subroutine linkage is appropriately established, the various subroutines can be used repeatedly.

A division instruction is not available as a hardware feature of the D17B. This deficiency can be remedied either software- or hardware-wise. The Von Neumann division subroutine has been developed in the TUSL for this purpose. The subroutines listed in this chapter are available for various versions of input/output interfaces to MCUG members as part of the information exchange program.

Section 4.1 describes in detail a software addition to the overall D17B checkout procedure. Full documentation is included on this subroutine since it is believed to be of importance to all users of the D17B. The sections which follow give brief descriptions of the various other subroutines that have been developed in the TUSL. Since all MCUG members will not desire full documentation on each of these subroutines, additional information on these subroutines will be made available on an exchange basis.

4.1 Exhaustive D17B Diagnostic Program

Because of the D17B's multi-channel disc memory, it is not sufficient to check the functional capabilities of the computer in one area of memory if a complete software diagnostic test is desired. For this reason, a new phase has been added to the diagnostic and checkout procedure--the Exhaustive D17B Diagnostic Program (EDP). Once each functional capability has been

checked out in one area of memory, it is not necessary to check each of the capabilities in each memory location. The reason for this is that the execution of instructions is performed using fixed logic, and this logic is, in general, not dependent on the channel or sector location of the instruction to be executed. The operations which will be tested exhaustively by the EDP are the following.

1) The ability to store data in each location.
2) The ability to retrieve data from each location.
3) The ability to unconditionally transfer to each location.
4) The ability to execute a flagged instruction in each location. The next instruction sector after this flagged instruction must be correct.

These four operations include a test of all the location dependent hardware in the D17B including all channels of the disc memory. The EDP will use address modification to perform the exhaustive test. The controlling program will be loaded in channel 00 after the functional capabilities of this channel are diagnosed. Therefore, if trouble results, the controlling program can be eliminated as the source of the trouble. A flow chart for the EDP program is shown in Figure 4-1. The program is broken into two main portions. The first section loads all 128 sectors from channel 04 through channel 50 with their respective addresses. That is, channel 32, sector 105 would contain an octal 00003305 at the end of this portion of EDP. These values will be used as indicators later in the second portion of the program.

The D17B will halt after it has loaded all sectors with the proper data. Since each location contains its absolute address, the program may be halted at this point for diagnosis of display hardware. The second portion of the program uses address modification to place the group of three instructions, shown below, sequentially through the memory.

```
C, s + 0    CLA      C + 2, s + 1
C, S + 1    CLA*U    C + 2, s + 2
C, s + 2    TRA      Control Program
```

(Where C = .02 through 46 and s = 000 through 177)

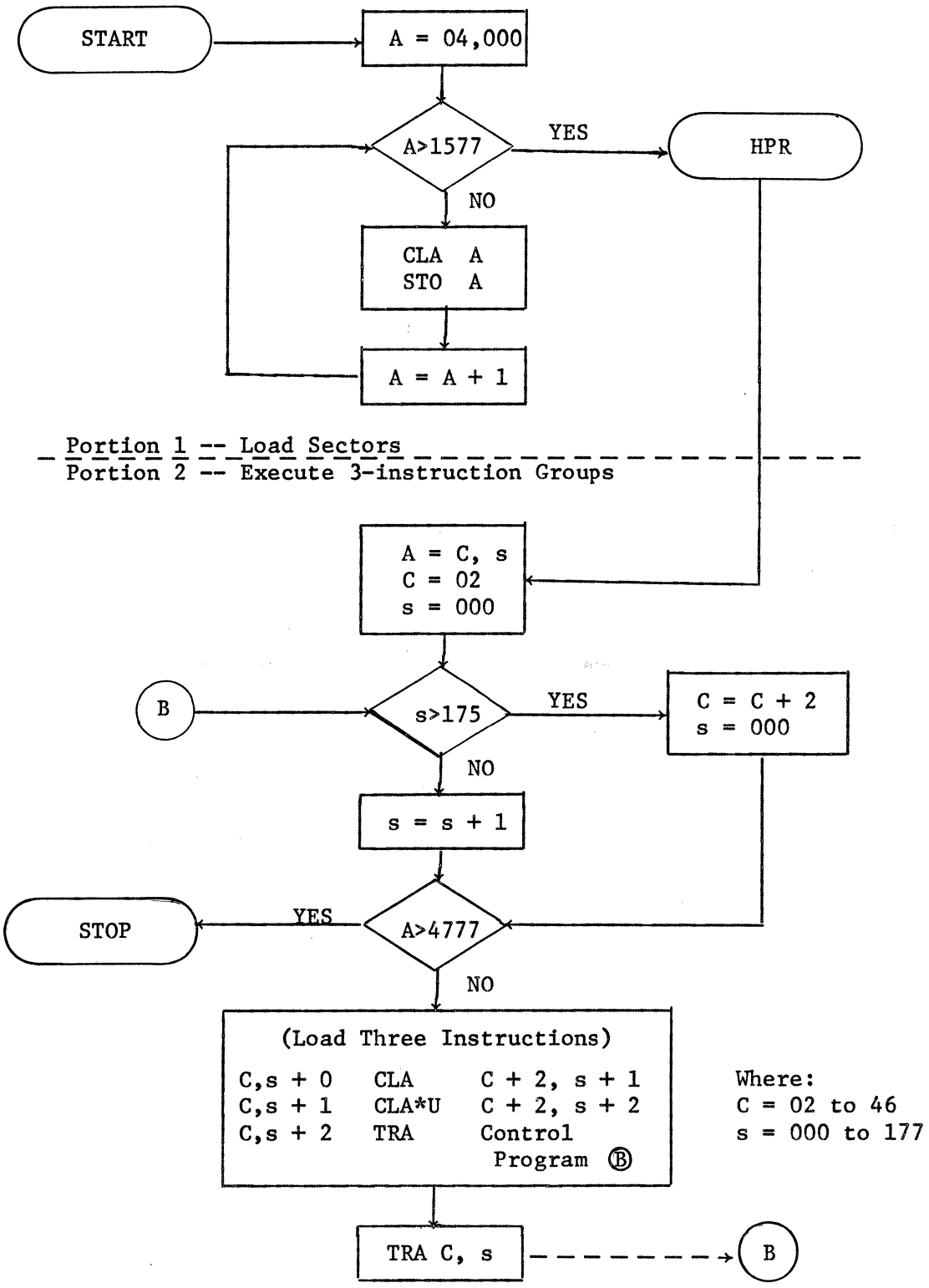


Fig. 4-1. Flowchart for exhaustive D17B diagnostic program (EDP).

It is important to note that each time these three instructions are stored in some group of sectors they will be executed immediately--before the next group of three is stored. This will allow the programmer to monitor the U-loop where he should observe a counter from 04,001 through 50,177. Since each group of three instructions is executed as it is stored, there will be a sufficient time delay between changes in the U-loop for the programmer to determine that each address (channel and sector) has been correctly flagged. The time delay is determined by the time required for the control program to construct the next group of three instructions, store them in the proper locations, and execute them. Another important point is that, at any one time during the second portion of EDP, the channel which is presently being used as the data channel will contain the groups of three instructions after the present instruction channel has been completely diagnosed.

Therefore, by placing these three instructions throughout the memory, each sector of every channel is checked for the capability to transfer to it from the control program, the capability to store data, the capability to retrieve data from it, the capability to execute a flagged instruction, and the ability to return to the controlling program. Thus, what started out to be a flagged diagnostic program has proven its ability to test much more than that, and it, therefore, deserves its title of EDP of the D17B Computer Diagnostic Testing Procedure. This program has been used to detect and assist in correcting problems associated with the TRA and TMI instructions and with external display equipment.

The program listing for EDP is shown in Table 4-1. No flagged instruction is used in the controlling program. One of the major functions of EDP is to test the flagged instructions; therefore, the controlling program should not contain a flagged instruction because these are under test.

Portion 1. Load sectors with addresses 0200-5177.

Location	Contents	Operation
0000	44020001	CLA 0001
0001	00000400	Data
0002	54040005	STO 0003
0003	00000000	Data
0004	44060005	CLA 0005
0005	00005177	Data
0006	74070003	SUB 0003
0007	10100021	TMI 0021
0010	44120011	CLA 0011
0011	54160002	Data
0012	64130003	ADD 0003
0013	54140017	STO 0015
0014	44150003	CLA 0003
0015	54000000	Data
0016	64200017	ADD 0016
0017	00000001	Data
0020	54040005	STO 0003
0021	44220022	CLA 0022
0022	40232200	HPR 2200

Portion 2. Place groups of three instructions from 0200-4777.

0023	44250024	CLA 0024
0024	00000177	Data
0025	54270030	STO 0026
0026	00000000	Data
0027	44300026	CLA 0026
0030	64320031	ADD 0031
0031	00000001	Data
0032	55160030	STO 0026
0116	01172221	ALS 2221
0117	01203221	ARS 3221
0120	11230121	TMI 0121
0121	75230122	SUB 0122
0122	77777600	Data
0123	75250124	SUB 0124
0124	00000174	Data
0125	11270126	TMI 0126
0126	44330026	CLA 0026
0127	45300026	CLA 0026
0130	01313207	ARS 3207
0131	01322207	ALS 2207
0132	65340133	ADD 0133
0133	00000200	Data
0134	54330030	STO 0026
0033	44350034	CLA 0034
0034	00004777	Data
0035	74360026	SUB 0026

Table 4-1. Exhaustive D17B diagnostic program (Sheet 1 of 2).

Location	Contents	Operation
0036	10370114	TMI 0114
0037	44410040	CLA 0040
0040	54670002	Data
0041	64420026	ADD 0026
0042	54430070	STO 0066
0043	44450044	CLA 0044
0044	55030003	Data
0045	64460026	ADD 0026
0046	54470104	STO 0102
0047	44510050	CLA 0046
0050	55060004	Data
0051	64520026	ADD 0026
0052	54530107	STO 0105
0053	44540026	CLA 0026
0054	64560055	ADD 0055
0055	00000001	Data
0056	00572221	ALS 2221
0057	00603205	ARS 3205
0060	10630061	TMI 0061
0061	74630062	SUB 0062
0062	76000000	Data
0063	64640026	ADD 0026
0064	64660065	ADD 0065
0065	44000201	Data
0066	54670000	STO XXXX
0067	44700026	CLA 0026
0070	64720071	ADD 0071
0071	00000002	Data
0072	00732224	ALS 2224
0073	00743210	ARS 3210
0074	10770075	TMI 0075
0075	74770076	SUB 0076
0076	77600000	Data
0077	65000026	ADD 0026
0100	65020101	ADD 0101
0101	47600202	Data
0102	55030000	STO XXXX
0103	45050104	CLA 0104
0104	50000027	Data
0105	55060000	STO XXXX
0106	45100107	CLA 0107
0107	50000000	Data
0110	65110026	ADD 0026
0111	55120115	STO 0113
0112	45130113	CLA 0113
0113	50000000	TRA XXXX
0114	45150115	CLA 0115
0115	41162200	HPR 2200

Table 4-1. Exhaustive D17B diagnostic program (Sheet 2 of 2).

The first portion of the test occupies sectors 000 through 022 of channel 00. This portion causes all channels to be loaded as discussed previously. The execution of this portion takes approximately three minutes. The second portion of EDP is loaded in sectors 023 through 134 of channel 00. This portion causes the groups of three instructions to be loaded and then transfers execution to each group as it is loaded. Execution time for this portion is approximately 12 minutes. The total time for execution of EDP is approximately 15 minutes. If the operator notices that the U-loop stops counting or if it counts in the wrong order, he should abort the program, find the present contents of the address counter, and begin to narrow the problem down under Single Cycle control in the range of locations where the problem occurred.

The combined use of functional checkout and EDP should provide the D17B status at any given time. If the D17B can execute EDP without an error, it is highly probable that the computer is completely in a "go" condition.

4.2 8-bit VOA Character Output

This subroutine sends an 8-bit data word to the Voltage Output A (VOA) register and causes the output device to print. The subroutine can transfer any size data block from any memory area to the output device. In general, it may also be used to transfer any number of 8-bit parallel words to an external or peripheral I/O device.

4.3 Octal Contents Output

This subroutine causes the peripheral I/O device to print the octal contents of the specified memory location. It is optional whether the sign will be printed.

4.4 Memory Dump Routine

In programming the D17B it is often convenient to obtain the actual contents of some group of locations in memory. This program will cause a block of data from any portion of memory to be printed. Both the location and the contents of each location will be printed. The printout is achieved using the VOA subroutine described in Section 4.2. The location and size of the block are entered using an interactive technique. This technique is accomplished using the Interactive Input (INI) module described in Section 3.3.

4.5 Waveform Generation

Digitized representations of analog waveforms are prestored in the D17B memory. These data are then brought to the voltage output register and sent to one of the D-A converters at the proper time.

4.6 Linear Interpolation

This subroutine linearly interpolates between specific data values to obtain the value corresponding to some unknown data. The required division operation in this subroutine is achieved by using the Von Neumann division subroutine.

4.7 Linear Scaling

This subroutine scales a series of data linearly by applying linearly weighted increments to these data.

4.8 Accumulator N-bit Rotation

This subroutine causes the contents of the accumulator to rotate right/left for n bits. Here rotation means that the contents are shifted while the information is preserved. The variable n can be any number.

4.9 Accumulator Bit Reversal

This subroutine reverses the contents of the accumulator bit-by-bit (i.e. A24 → A1, A23 → A2, etc.) and preserves the original data order in the lower accumulator.

4.10 Logic EXCLUSIVE OR Operation

The corresponding bits of the two specified memory locations are logically EXCLUSIVE ORed. The result is stored in the accumulator.

4.11 Binary-to-BCD Conversion

This subroutine converts any 10-bit binary number (0000 to 1747 octal) to the 12-bit BCD equivalent (000 to 999 decimal). The resulting decimal number can be printed on an output device.

4.12 BCD-to-Octal Conversion

This subroutine converts a Binary Coded Decimal number to its octal equivalent. The maximum BCD number allowed is 999999.

4.13 Von Neumann Division

A division instruction is not available as a standard hardware feature on the D17B. An efficient division subroutine has been developed in which the Von Newmann algorithm has been applied. The programmer has to assure by scaling if necessary that the absolute value or magnitude of the divisor is greater than that of the dividend. The quotient is stored in the accumulator.

4.14 Numerical Integration

This subroutine integrates the area under the digitized representation of an analog waveform which is inputed to the D17B using discrete input instructions. This subroutine may also be used to integrate the representation of a waveform using data values that have been previously stored in memory. The trapezoidal rule of numerical integration is used.

4.15 Factorials

This subroutine calculates the factorial of any constant n. These numbers are treated as integers, and the result is stored in the accumulator.

4.16 Reciprocal of a Constant

The reciprocal of a constant is calculated by using this subroutine. An iterative algorithm which calculates the next value from the previous value is used. Both positive and negative numbers are permissible.

4.17 Sin/Cos Routines

Separate subroutines are available for calculating Sin (x) and Cos (x). Series expansions of the respective trigonometric functions are used. The argument (x) is limited to the range of -1 to +1 radian. A subroutine which calculates Sin(x) and Cos(x) in the split word format is also available. A nesting technique is used in calculating the series throughout these examples.

4.18 Negative Natural Exponential (e^{-x})

This subroutine calculates the negative natural exponential function by using the series expansion. The positive exponential function can be obtained by calculating the reciprocal. x must be less than unity.

4.19 Natural Logarithm ln(x)

This subroutine calculates the natural logarithmic function. The value of the argument (x) is limited to positive values less than two.

4.20 Floating Point Multiply

This subroutine will multiply two floating point numbers. The two mantissas are multiplied, and the exponents are added. Both full- and split-word versions are available.

4.21 Floating Point ADD/SUB

A provision for floating point operations helps to give the D17B more complete arithmetic capability. Proper scaling is provided in the split-word format.

4.22 Statistical Mean and Variance

This subroutine calculates the mean and/or variance of an array of data. The size of the array is variable and need not be known by the programmer.

4.23 Fast Fourier Transform (FFT)

This subroutine transforms a time-domain representation into its equivalent in the frequency domain and vise versa. The time series can be formed using a maximum of 128 data values. Extensions are under development.

4.24 Interactive Arithmetic

This subroutine can perform addition, subtraction, multiplication, and division through interactive communication between the D17B and the user.

4.25 Optimization

This subroutine uses the steepest descent algorithm to implement an efficient optimization calculation.

CHAPTER 5

CORRECTIONS TO THE D17B PROGRAMMING MANUAL

Several additions and corrections to the D17B Programming Manual, Report MCUG-4-71, will be presented in this chapter. These additions and corrections are listed corresponding to their page numbers as follows:

Page	Line	Addition/Correction
13	5	... the operand sector address. *Also, in the case of a TRA instruction, both the next instruction sector address and the sector of the designated flag loop will be determined by the operand sector address.*
29	15	If (c) is 50, F, E, or H, only L *or U* may be used for flag storing.
29	27	SMP Split Multiply 20 c,s 7 wdt $(A)_{1-11} \longrightarrow (L)_{14-24}$ and $(A)_{14-24} \longrightarrow (L)_{2-12}$ *Note that the right-half split word of (L) is shifted left by one bit position.* $(A)_{14-24} \cdot (m)_{14-24} \longrightarrow (A)_{14-24}$ and $(A)_{1-11} \cdot (m)_{1-11} \longrightarrow (A)_{1-11}$
31	19	... result stored in (A). *(L) remains unchanged after execution.*
31	36	$(A)_{20}$ *should be* $(A)_{21}$
31	37	$(A)_{21}$ *should be* $(A)_{20}$
32	5	BCD *should be binary*
32	18	BOA Binary Output A 40 10,s 1 wdt *The Binary Output A flip-flop is* G_1. *If* G_{11} *and* G_{10} *are 0, a 1 is added to* $(A)_{17-24}$. *If* G_{11} *is a 0 and* G_{10} *is a 1, a 1 is subtracted from* $(A)_{17-24}$. *The new* G_{11} *and* G_{10} *are determined from:* $(A)_{24} = 0 \longrightarrow G_{11} = 0,\ G_{10} = 1;\ (A)_{24} = 1 \longrightarrow G_{11} = 1,\ G_{10} = 0$