Datapoint 2200

PROGRAMMERS' MANUAL

A product of COMPUTER TERMINAL CORPORATION
9725 Datapoint Drive
San Antonio, Texas 78229

DATAPOINT 2200

PROGRAMMERS' MANUAL

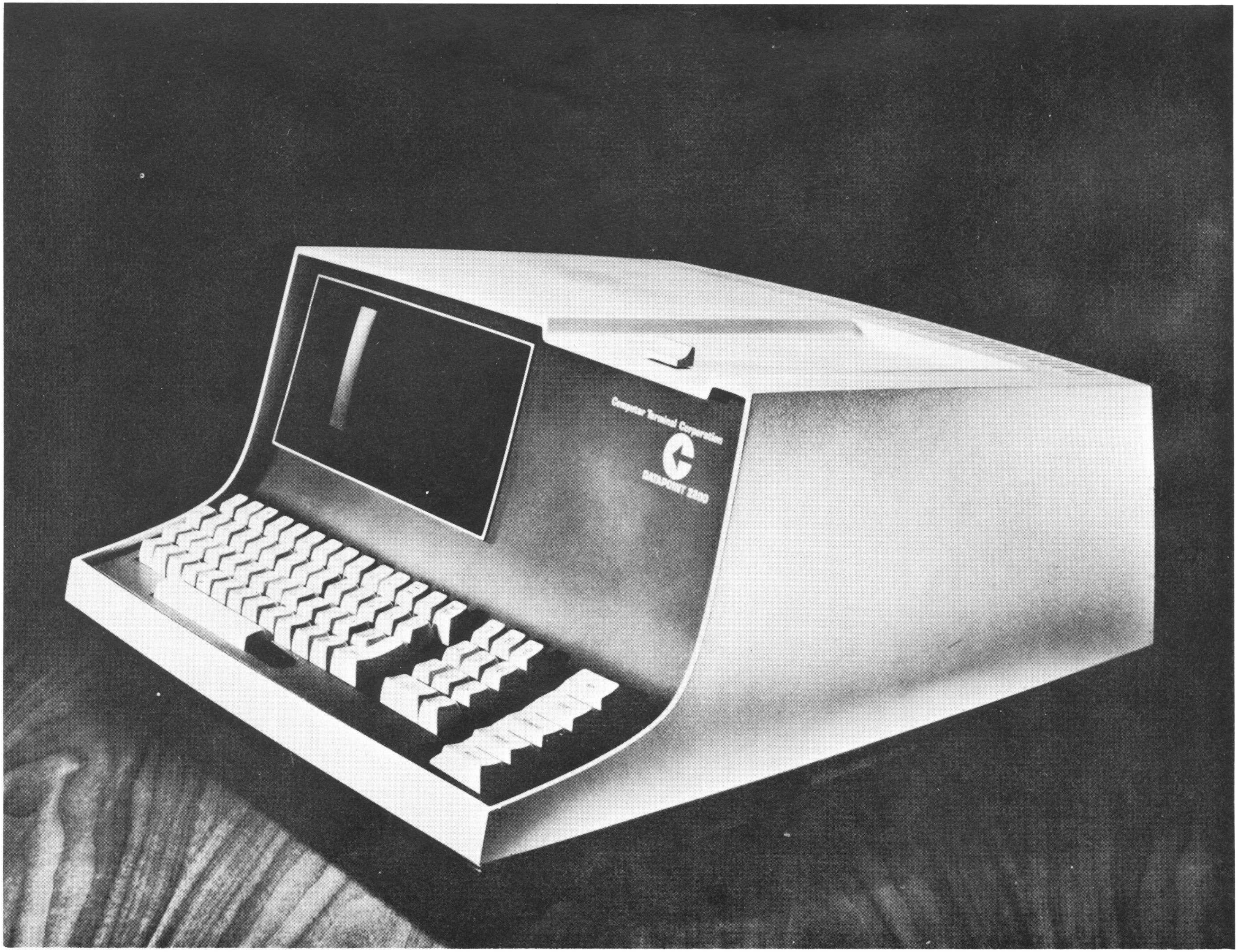
August, 1971

COMPUTER TERMINAL CORPORATION

9725 Datapoint Drive

San Antonio, Texas 78229

TABLE OF CONTENTS


Computer Terminal Corporation
DATAPOINT 2200

SECTION 1

DATAPOINT 2200

REFERENCE MANUAL

PART 1

GENERAL FEATURES

1.1 INTRODUCTION

The Computer Terminal Corporation Datapoint 2200 is an integrated data system consisting of an alphanumeric keyboard for data entry, a cathode ray screen for data display, two digital cassette recorders for bulk data storage, a general purpose digital computer for control, and a communications capability for interface with external devices and communications facilities.

Through programming of the control computer the Datapoint 2200 may be used for an infinite varity of data processing applications.

The achievement of a small computer with integrated keyboard, display, storage and communications at such low cost now makes possible computer sophistication for applications not previously practical - particularly in the computer terminal/data entry/communications area.

This manual describes the specific hardware details of the Datapoint 2200. For information regarding specific applications the Datapoint 2200 Programmers' Guide and specific application manuals should be referred to.

1.2 SYSTEM ELEMENTS

There are four basic system elements in the basic Datapoint 2200 plus the capability of interface to a number of external perpherial devices.

This manual covers the basic elements (c.r.t., keyboard, processor, cassette tape decks) and one external device (communications adaptor).

1.3 CRT DISPLAY

The Datapoint 2200 CRT Display provides the following features:

a. 7" x 2-1/2" viewing area;

b. 960 characters;

c. 80 character by 12 line format;

d. 4/32" x 3/32" character size;

e. Entire 94 character ASCII set;

f. 60 frame per second refresh rate;

g. 5 x 7 matrix character generation;

h. 5 x 7 solid, blinking cursor, alternates with character, nondestructive;

i. P31 green phosphor;

j. Single control line erasure, frame erasure, and page roll-up; and

k. Direct control of all c.r.t. functions by the 2200 processor, providing tab, editing, form control, etc.

1.4 KEYBOARD

The integral keyboard provides a basic 41 key alphanumeric key group, an 11 key numeric group and five system control keys.

The keyboard provides a unique multi-key roll-over characteristic providing maximum ease of typing. Transfer of characters from the keyboard is under control of the 2200 processor. An audible click providing an acoustical feedback to the typist is available under processor control.

A programmable audio "beep" is also provided when it is desired to gain a typist's attention.

1.5 PROCESSOR

The integral processor provides all control functions and includes:

a. 28 different instruction types;

b. 7 addressable registers;

c. 7 deep pushdown stack;

d. 8 bit memory word length;

e. Up to 8192 word memory;

f. Complete parallel I/O system;

g. Automatic power-up restart.

1.6 CASSETTE TAPE DECKS

Two read-write tape decks are provided for program and data storage. The deck accepts Norelco-type cassettes and provides:

a. 47 characters per inch density;

b. Dual-capstan forward-reverse operation;

c. Processor controlled data transfer, direction control, and high-speed rewind.

1.7 COMMUNICATIONS ADAPTOR

The communications adaptor is a unique feature of the Datapoint 2200 system. There are four versions of the adaptor:

a. EIA RS-232 interface for use with external data sets or peripherals;

b. High-level keying interface for connection to telegraph-type communications channels or equipment;

c. 103-type data set interface for direct connection to common carrier lines, and including automatic dialing and answering;

d. 202-type data set interface with 150 bit/sec supervisory channel operation for direct connection to common carrier lines, and including automatic dialing and answering.

The adaptor permits program selection of the desired bit rate, character length, and character set providing the most versatile communications capability yet provided for a remote terminal.

1.8 GENERAL SPECIFICATIONS

The Datapoint 2200 has the following general characteristics:

a. 105-135 v.a.c., 60 cycle, 180 watts, power input;

b. 47 pounds weight;

c. 9-5/8" high, 18-1/2" wide, by 19-5/8" deep outside dimensions;

d. 0° to 50°C (32° to 122°F), 10 to 90 percent relative humidity operation environment.

1.9 OPTIONAL PERIPHERALS

A number of optional peripherals are available (in addition to the communications adaptor) for use with the Datapoint 2200 including a:

a. 132 column, 30 c.p.s. impact page printer; and a

b. IBM compatible magnetic tape deck.

For further information on these devices reference should be made to their respective reference manuals.

PART 2

BASIC PROCESSOR

2.1 PROCESSOR ORGANIZATION

The processor contained in the Datapoint 2200 is comprised of the Arithmetic/Logic Unit, 7 program accessible registers, 2K to 8K words of read/write memory, an instruction decoder and a seven-level hardware pushdown stack used in subroutine type operations.

2.2 ARITHMETIC/LOGIC UNIT

The Arithmetic/Logic Unit is capable of processing both binary integers and logical operands. All arithmetic and logical operations may take place between the A-register and any of the 7 program accessible registers (or between the A-register and memory). The A-register always contains the result of an arithmetic or logic operation, with the other register (or memory cell) being unaffected. Arithmetic and logic operations affect the Sign, Carry, Zero and Parity Flip-flops.

2.3 PROCESSOR REGISTERS

A - The Accumulator register is used to hold the result of all arithmetic and logical instructions. All data transfers into or out of the computer take place through this register.

7	6	5	4	3	2	1	0	A Register

B, C, D, E - These are general purpose registers which may be used in conjunction with the Accumulator in arithmetic and logical operations. Each register may be loaded from or stored into memory or another register. When used in conjunction with the A and H, L registers, the B, C, D and E registers may function as indexes.

7	6	5	4	3	2	1	0	B, C, D, E Registers

H, L - The H and L Registers are utilized to contain respectively the most significant portion (MSP) and least significant portion (LSP) of the address of a memory location being referenced. All memory reference instructions utilize these registers with the exception of CALL and JUMP commands. However, the H and L Registers may be used as general purpose registers when not being used as above.

H (8 bits)	L (8 bits)
MSP of address being referenced	LSP of address being referenced

P - The program register or "Location Counter" contains the address of the next instruction to be executed. This register is stored in the pushdown stack upon the execution of a "CALL" instruction and is loaded with the effective address upon execution of a "JUMP", "CALL" or "RETURN" instruction. The P register is 13 bits in length and is capable of addressing up to 8K of memory.

12	P (13 bits)	0

I - The I register is the register which holds the "operation code" of the instruction currently being executed. The contents of I are gated through a decoding network to determine what operation,internal or external, is to be performed.

2.4 MEMORY

The basic Datapoint 2200 is supplied with 2048 eight-bit words of memory. Additional modules of 2048 words each may be incorporated with the total memory capacity of the processor being 8192 words. Each 2K memory is made up of 32 individual MOS shift registers with each one having a capacity of 512 bits or 64 eight-bit words. These registers are clocked at a rate of 1.2 MHz. Data is read out in bit serial fashion with one word taking 8 microseconds. During this period of time, two clock pulse times are available for the processor to perform any necessary gating or testing functions.

The Datapoint 2200 memory might be likened to a drum type memory in some respects. It takes approximately 1/2 millisecond for the memory to completely circulate. Thus, if the current instruction referenced a memory location for data access, there would be a 1/2 millisecond delay before that instruction could be completed. However, unlike a drum memory the MOS memory may be stopped during instruction execution so that each succeeding instruction may be read from memory without delay (in 8 usec.).

Physically, instructions require a variable number of cycles for completion. In the first cycle, the instruction is fetched from memory and decoded. If the instruction involves no memory reference, it is then executed within 8 microseconds for a total completion time of 16 microseconds.

"Immediate" type instructions are the same as instructions requiring no memory reference and require a 16 usec interval for the operand fetch and execute cycle. Jump and Call type instructions require a variable amount of time for execution, depending on the difference between the old and new locations.

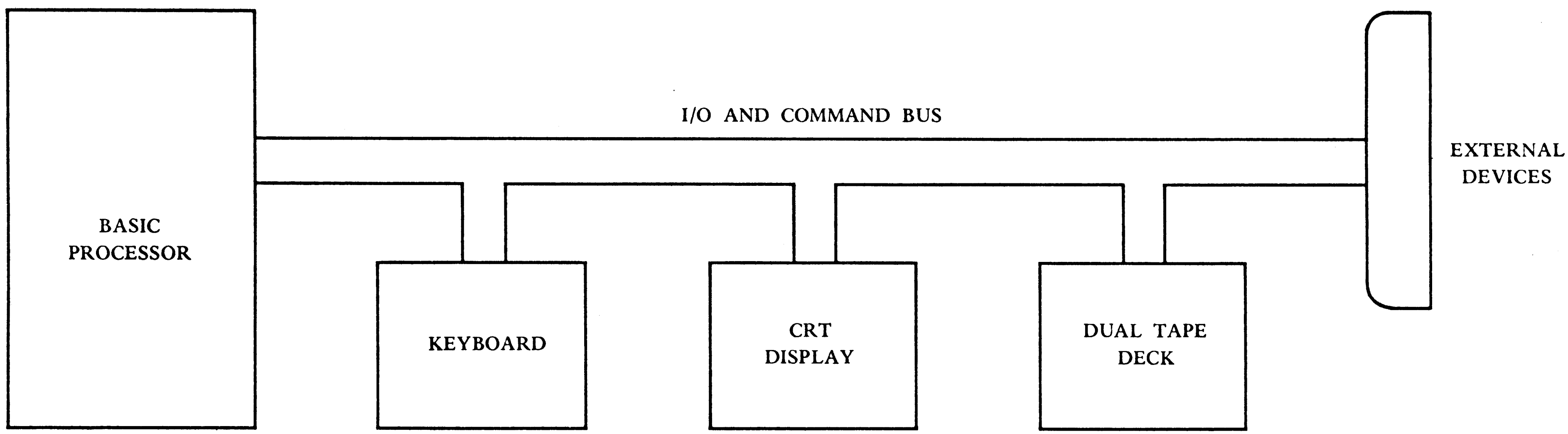


FIGURE 2-1

DATAPOINT 2200
BLOCK DIAGRAM

2.5 PUSHDOWN STACK

A unique feature of a machine this size is the incorporation into the processor's structure of a pushdown stack which is useful in any type of application which requires program subroutines. The stack automatically stores the contents of the P register upon execution of a "CALL" instruction and automatically restores P upon execution of a "RETURN". The stack is a group of "last-in/first-out" registers and has a capacity of 7 CALLS. Note that "CALLS" may be "nested", that is more than one CALL may be made before the execution of a RETURN. The execution of a "RETURN" will cause processor control to be given to the next instruction following the last executed CALL.

Pushdown Stack

Address of CALL 5	
Address of CALL 4	Maximum
Address of CALL 3	Capacity of
Address of CALL 2	7 CALLS
Address of CALL 1	

13 Bits

2.6 CONTROL FLIP-FLOPS

Also contained in the basic processor are four control flip-flops which reflect the state of the arithmetic logic unit and which may be tested through the execution of a conditional jump, call or return instruction. The flip-flop mnemonics with their associated functions are as follows:

C_f-Carry Flip-flop. Set when an arithmetic operation results in either a carry (add) or borrow (subtract).* The Carry Flip-flop also reflects the state of the most significant bit in the accumulator after completion of a shift right instruction. Likewise, it reflects the state of the accumulator least significant bit after completion of a shift left instruction.

Z_f-Zero Flip-flop. Set when the result of an arithmetic or logical operation is equal to zero.*

S_f-Sign Flip-flop. This flip-flop reflects the state of bit 7 in the accumulator after an arithmetic type operation.*

P_f-Parity Flip-flop. Indicates the parity or "number of one bits" contained in the accumulator. If this flip-flop is set (true), the A register contains an odd number of one bits; if it is reset (false), the A register contains an even number of one bits.*

*In the event of a compare instruction the contents of the accumulator are not changed; however, the control flip-flops reflect the equivalent of a subtract instruction.

2.7 DATA FORMAT

Data is represented in the Datapoint 2200 in the form of 8-bit binary integers.

7	6	5	4	3	2	1	0

DATA WORD

2.8 INSTRUCTION FORMATS (GENERAL)

Instruction formats, dependent upon the operation to be performed, may be eight, sixteen or twenty-four bits in length.

Type-1- register to register, memory reference, arithmetic, logical, shift instructions

OP CODE 8 bits

Type-2- immediate mode instructions

OP CODE 8 bits	OPERAND 8 bits

Type-3- JUMP & CALL instructions

OP CODE 8 bits	ADDRESS 16 bits

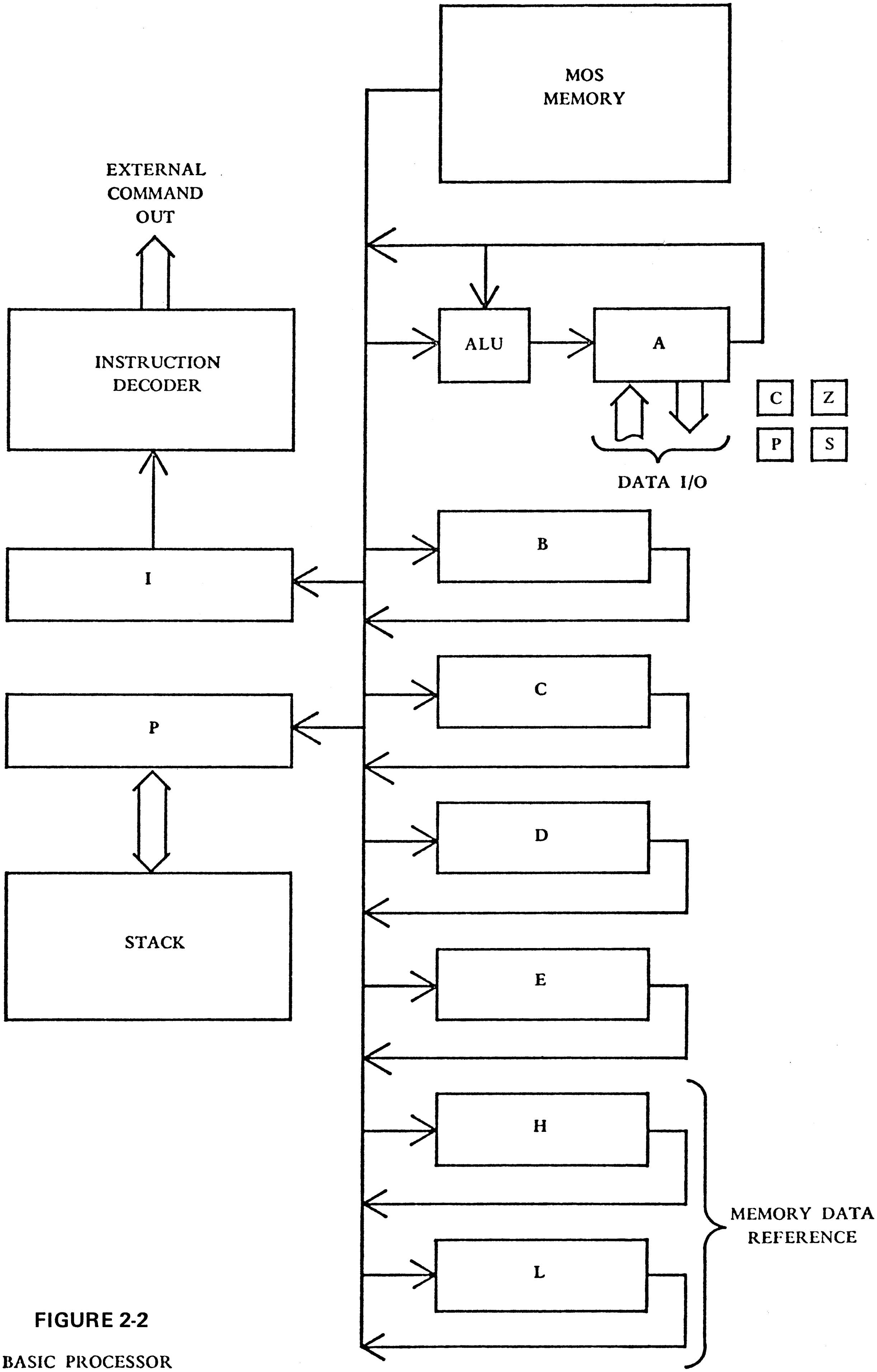


FIGURE 2-2

BASIC PROCESSOR

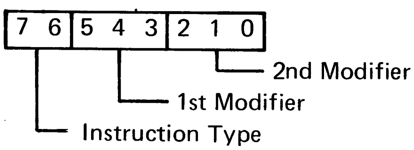
PART 3

INSTRUCTION REPERTOIRE

3.1 PRESENTATION FORMAT

This section gives a detailed description of each of the Datapoint 2200 instructions. The use and operations of each instruction is presented as follows:

FUNCTION:	Mnemonic Code
OPERATION:	Symbolic representation of instruction description.
OP CODE:	Operation Code, expressed in octal.
TIMING:	Execution time. (Times are approximate).
DESCRIPTION:	Definition of function of the instruction.



INSTRUCTION FORMAT: Explanation of the function of each part of the instruction word.

NOTE

Considerations in instruction use and further definition of function.

Symbols and Abbreviations

The following symbols and abbreviations are used in the instruction format:

( )	the contents of
←	is replaced by
→	is transferred to
:	is compared with
A	8 bit arithmetic register (accumulator)
B, C, D, E	8 bit general purpose registers
H	8 bit register used to specify most significant portion of operand address
L	8 bit register used to specify least significant portion of operand address
M	memory location designated by contents of H, L
r	one of the following register designators: A, B, C, D, E, H, L
r_s	designates operand source register (s=0-7)
r_d	designates operand destination register (d=0-7)
$\vee$	Logical "OR" operation
$\veebar$	Logical "exclusive-OR" operation
$\wedge$	Logical "AND" operation
STACK	Instruction counter (P) pushdown queue
P	Program counter
f_c	Flag flip-flop codes: C_f, Z_f, S_f, P_f
RR	Register to Register
IM	Immediate (from P+1)
MR	Memory Reference (Contents of memory location designated by H, L)
I	Instruction Register

TABLE 3-1

SOURCE AND DESTINATION CODES (s and d)

	s/d	SYMBOLIC CODE	
r_s/r_d	0	A	A Register
	1	B	B Register
	2	C	C Register
	3	D	D Register
	4	E	E Register
	5	H	H Register
	6	L	L Register
M	7	M	Memory location specified by contents of H&L

TABLE 3-2

FLIP-FLOP CODE (f_c)

c	SYMBOLIC CODE	NAME
0	C_f	Carry
1	Z_f	Zero
2	S_f	Sign
3	P_f	Parity

LOAD IMMEDIATE: Lr_d

OP CODE: 0d6 TIMING: 16 usec.

OPERATION: $(P+1) \rightarrow r_d$, $P+2 \rightarrow P$

DESCRIPTION: Transfers the contents of the memory location immediately following the instruction, to the register specified by bits 3-5(d) of the instruction.

INSTRUCTION FORMAT:

P			P+1
7 6	5 4 3	2 1 0	7 0
0	d	6	OPERAND

d: is the destination designator
d=7: is not allowed

NOTE

1. The contents of P+1 are unchanged.
2. None of the Flag Flip-flops are affected.
3. Refer to Table 3-1 for destination codes.

LOAD: Lr_dM, Lr_dr_s, LMr_s

OP CODE: 3ds

TIMING: 16 usec. for register to register transfers, 520 usec. for memory reference.

OPERATION: $(M) \rightarrow r_d$ s=7, d≤6 (Lr_dM)
$(r_s) \rightarrow r_d$ s≤6, d≤6 (Lr_dr_s)
$(r_s) \rightarrow M$ s≤6, d=7 (LMr_s)

DESCRIPTION: Transfers the operand from the source specified by bits 0-2 of the instruction word to the destination specified by bits 3-5 of the instruction word.

INSTRUCTION FORMAT:

P		
7 6	5 4 3	2 1 0
3	d	s

d: designates the destination of data.
s: designates the source. If either s or d=7 a memory reference is indicated and the contents of registers H&L specify the address of the memory location.

NOTE

1. The data source is unaffected.
2. s & d both = 7 results in a Halt instruction.
3. None of the Flag Flip-flops are affected by execution of this instruction.
4. s=d results in a NOP, except as stated in Note 2.

ADD IMMEDIATE: AD

OP CODE: 004 TIMING: 16 usec.

OPERATION: (A) + (P+1) → A, P+2 → P

DESCRIPTION: Adds to the contents of the A register the contents of the memory location immediately following the instruction, and retains the sum in the A register. Sets the C_f Flip-flop if ADD overflow occurs, otherwise resets C_f.

INSTRUCTION FORMAT:

P			P+1
7 6	5 4 3	2 1 0	7 0
0	0	4	OPERAND

NOTE

1. The Sign, Zero and Parity Flip-flops will indicate the status of the A register at completion.
2. The contents of P+1 are unchanged.
3. The Carry Flip-flop is cleared at the beginning of this instruction.

ADD WITH CARRY IMMEDIATE: AC

OP CODE: 014 TIMING: 16 usec.

OPERATION: (A) + (P+1) + (C_f) → A, P+2 → P

DESCRIPTION: Adds the C_f bit and the contents of the location immediately following the instruction to the contents of the A register, and retains the sum in the A register. If add overflow occurs, the C_f Flip-flop is set, otherwise C_f is reset.

INSTRUCTION FORMAT:

P			P+1
7 6	5 4 3	2 1 0	7 0
0	1	4	OPERAND

NOTE

1. The Sign, Zero and Parity Flip-flops will indicate the status of the A register at completion.
2. The contents of P+1 remain unchanged.

ADD: ADr_s ADM

OP CODE: 20s TIMING: 16 usec. if RR, 520 usec. if MR

OPERATION: (A) + (r_s) → A or (A) + (M) → A

DESCRIPTION: This instruction is identical to ADD IMMEDIATE with the exception of operand source.

INSTRUCTION FORMAT:

P		
7 6	5 4 3	2 1 0
2	0	s

s: specifies the operand source. Refer to Table 3-1 for source codes.

ADD WITH CARRY: ACr_s ACM

OP CODE: 21s TIMING: 16 usec. if RR, 520 usec. if MR

OPERATION: (A) + (C_f) + (r_s) → A or (A) + (C_f) + (M) → A

DESCRIPTION: This instruction is identical to ADD WITH CARRY IMMEDIATE with the exception of operand souce.

INSTRUCTION FORMAT:

P		
7 6	5 4 3	2 1 0
2	1	s

s: specifies the operand source. Refer to Table 3-1 for source codes.

SUBTRACT IMMEDIATE: SU

OP CODE: 024 TIMING: 16 usec.

OPERATION: (A) - (P+1) → A, P+2 → P

DESCRIPTION: Subtracts the contents of the memory location immediately following the instruction from the contents of the A register, and retains the difference in the A register. The C_f Flip-flop is set if underflow occurs.

INSTRUCTION FORMAT:

P			P+1
7 6	5 4 3	2 1 0	7 0
0	2	4	OPERAND

NOTE

1. The contents of P+1 is unchanged.
2. The Zero, Sign, and Parity Flip-flops represent the status of the A register at the completion of this instruction.

SUBTRACT WITH BORROW IMMEDIATE: SB

OP CODE: 034 TIMING: 16 usec.

OPERATION: (A) - (P+1) - (C_f) → A, P+2 → P

DESCRIPTION: Subtracts the contents of the memory location immediately following the instruction and the C_f bit, from the contents of the A register. Sets the C_f bit if underflow occurs, otherwise resets C_f.

INSTRUCTION FORMAT:

P			P+1
7 6	5 4 3	2 1 0	7 0
0	3	4	OPERAND

NOTE

1. The contents of P+1 are unchanged.
2. The Zero, Sign, and Parity Flip-flops represent the status of the A register at the completion of this instruction.

SUBTRACT: SUr_s SUM

OP CODE: 22s TIMING: 16 usec. if RR, 520 usec. if MR

OPERATION: (A) - (r_s) → A or (A) - (M) → A

DESCRIPTION: This instruction is identical to SUBTRACT IMMEDIATE with the exception of operand source.

INSTRUCTION FORMAT:

P		
7 6	5 4 3	2 1 0
2	2	s

s: specifies the operand source. Refer to Table 3-1 for source codes.

SUBTRACT WITH BORROW: SBr_s SBM

OP CODE: 23s TIMING: 16 usec. if RR, 520 usec. if MR

OPERATION: (A) - (r_s) - (C_f) → A or (A) - (M) - (C_f) → A

DESCRIPTION: This instruction is identical to SUBTRACT WITH BORROW IMMEDIATE with the exception of operand source.

INSTRUCTION FORMAT:

P		
7 6	5 4 3	2 1 0
2	3	s

s: specifies the operand source. Refer to Table 3-1 for source codes.

AND IMMEDIATE: ND

OP CODE: 044 TIMING: 16 usec.

OPERATION: (P+1) ∧ (A) → A, P+2 → P

DESCRIPTION: Forms the logical product of the contents of the A register with the contents of the memory location immediately following the instruction, and places the results in the A register.

INSTRUCTION FORMAT:

P			P+1
7 6	5 4 3	2 1 0	7 0
0	4	4	OPERAND

NOTE

1. The Carry Flip-flop will be reset upon completion of the operation.
2. The Zero, Sign, and Parity Flip-flops will represent the status of the A register upon completion of the operation.

SAMPLE OPERATION:

(A Reg)	0	0	1	1
(P+1)	0	1	0	1
(A Reg)	0	0	0	1

OR IMMEDIATE: OR

OP CODE: 064 TIMING: 16 usec.

OPERATION: (A) V (P+1) → A, P+2 → P

DESCRIPTION: Forms the logical sum of the contents of the A register and the contents of the memory location immediately following the instruction, and places the result in the A register.

INSTRUCTION FORMAT:

P			P+1
7 6	5 4 3	2 1 0	7 0
0	6	4	OPERAND

NOTE

1. The Carry Flip-flop will be reset at conclusion.
2. The Zero, Sign, and Parity Flip-flops will represent the status of the A register at completion of the operation.

SAMPLE OPERATION:

(A Reg)	0	0	1	1
(P+1)	0	1	0	1
(A Reg)	0	1	1	1

AND: NDr_s, NDM

OP CODE: 24s TIMING: 16 usec. if RR, 520 usec. if MR

OPERATION: (A) ∧ (r_s) → A, or (A) ∧ (M) → A

DESCRIPTION: This instruction is identical to AND IMMEDIATE with the exception of operand source.

INSTRUCTION FORMAT:

7 6	5 4 3	2 1 0
2	4	s

s: specifies the operand source. Refer to Table 3-1 for source codes.

OR: ORr_s ORM

OP CODE: 26s TIMING: 16 usec. if RR, 520 usec. if MR

OPERATION: (A) V (r_s) → A, or (A) V (M) → A

DESCRIPTION: This instruction is identical to OR IMMEDIATE with the exception of operand source.

INSTRUCTION FORMAT:

P		
7 6	5 4 3	2 1 0
2	6	s

s: specifies the operand source. Refer to Table 3-1 for source codes.

EXCLUSIVE OR IMMEDIATE: XR

OP CODE: 054 TIMING: 16 usec.

OPERATION: $(A) \veebar (P+1) \rightarrow A, P+2 \rightarrow P$

DESCRIPTION: The logical difference of the contents of the A register and the contents of the memory location immediately following the instruction is formed, and the result replaces the contents of the A register.

INSTRUCTION FORMAT:

P			P+1
7 6	5 4 3	2 1 0	7 0
0	5	4	OPERAND

NOTE

1. The Carry Flip-flop will be reset at conclusion.
2. The Zero, Sign and Parity Flip-flops will represent the status of the A register upon completion of the operation.

SAMPLE OPERATION:

(A Reg)	0	0	1	1
(P+1)	0	1	0	1
(A Reg)	0	1	1	0

COMPARE IMMEDIATE: CP

OP CODE: 074 TIMING: 16 usec.

OPERATION: $(A) : (P+1), P+2 \rightarrow P$

DESCRIPTION: Compares the contents of the A register with the contents of the memory location immediately following the instruction. The flag flip-flops assume the same state as they would for a Subtract instruction.

INSTRUCTION FORMAT:

P			P+1
7 6	5 4 3	2 1 0	7 0
0	7	4	OPERAND

NOTE

1. The contents of the A register are unaffected.

EXCLUSIVE OR: XRr_s XRM

OP CODE: 25s TIMING: 16 usec. if RR, 520 usec. if MR

OPERATION: $(A) \vee (r_s) \rightarrow A, (A) \vee (M) \rightarrow A$

DESCRIPTION: This instruction is identical to EXCLUSIVE OR IMMEDIATE with the exception of operand source.

INSTRUCTION FORMAT:

P		
7 6	5 4 3	2 1 0
2	5	s

s: specifies the operand source. Refer to Table 3-1 for source codes.

COMPARE: CPr_s CPM

OP CODE: 27s TIMING: 16 usec. if RR, 520 usec. if MR

OPERATION: $(A) : (r_s)$ or $(A) : (M)$

DESCRIPTION: This instruction is identical to COMPARE IMMEDIATE with the exception of operand source.

INSTRUCTION FORMAT:

P		
7 6	5 4 3	2 1 0
2	7	s

s: specifies the operand source. Refer to Table 3-1 for source codes.

UNCONDITIONAL JUMP: JMP

OP CODE: 104 TIMING: Variable*

OPERATION: (P+1, P+2) → P

DESCRIPTION: An unconditional transfer of control. The contents of P+1 represent the least significant portion of the address, while the contents of P+2 represent the most significant portion.

INSTRUCTION FORMAT:

P			P+1	P+2
7 6	5 4 3	2 1 0	7 0	7 0
1	0	4	LSP	MSP
OP CODE			ADDRESS	

The three high order bits in the address are ignored, the remaining 13 bits specify the address to which control is to be transferred.

NOTE

*Timing is variable dependent upon cyclic difference between instruction and effective address locations.

JUMP IF CONDITION TRUE: JTf_c

OP CODE: 1(c+4)0 TIMING: Variable if condition true, 24 usec. if condition false.

OPERATION: If (f_c=TRUE), (P+1 , P+2) → P. Otherwise, P+3 → P.

DESCRIPTION: Examines the designated flip-flop. If set, transfers control to the address designated by the contents of the two memory locations immediately following the instruction. If the selected flip-flop is reset, executes the next sequentially available instruction.

INSTRUCTION FORMAT:

P			P+1	P+2
7 6	5 4 3	2 1 0	7 0	7 0
1	c+4	0	LSP	MSP
OP CODE			ADDRESS	

c: designates which flip-flop condition is to be tested. Refer to Table 3-2 for list of Flip-flop codes.

NOTE

1. The condition of the selected Flip-flop is unchanged by this instruction.

JUMP IF CONDITION FALSE: JFf$_c$

OP CODE: 1c0

TIMING: Variable if condition false, 24 usec. if condition true.

OPERATION: If (f_c=FALSE), (P+1, P+2) → P. Otherwise P+3 → P.

DESCRIPTION: Examines the designated flip-flop. If reset, transfers control to the address designated by the contents of the two memory locations immediately following the instruction. If the selected flip-flop is set, executes the next sequentially available instruction.

INSTRUCTION FORMAT:

P			P+1	P+2
7 6	5 4 3	2 1 0	7 0	7 0
1	c	0	LSP	MSP
OP CODE			ADDRESS	

c: designates which flip-flop (condition) is to be tested. Refer to Table 3-2 for list of flip-flop codes.

NOTE

1. The condition of the selected flip-flop is unchanged by this instruction.

SUBROUTINE CALL: CALL

OP CODE: 106

TIMING: Variable

OPERATION: P+3 → STACK, (P+1, P+2) → P

DESCRIPTION: Transfers the address of the next sequentially available instruction to the Pushdown Stack, and transfer control to the address specified by the contents of the two memory locations immediately following the Op Code.

INSTRUCTION FORMAT:

P			P+1	P+2
7 6	5 4 3	2 1 0	7 0 0	7 0
1	0	6	LSP	MSP
			ADDRESS	

NOTE

1. The Stack is open-ended in operation. If it is overfilled, the deepest address will be lost.

CONDITIONAL SUBROUTINE CALL

IF CONDITION TRUE: CTf_c

OP CODE: 1(c+4)2

TIMING: Variable if condition true, 24 usec. if condition false.

OPERATION: If (f_c=TRUE), P+3 → STACK, (P+1, P+2) → P. Otherwise, P+3 → P.

DESCRIPTION: Examines the designated flip-flop. If set, transfers the address of the next sequentially available instruction to the pushdown stack, and transfers control to the address of the two memory locations immediately following the Op Code. If the selected flip-flop is reset, executes the next sequentially available instruction.

INSTRUCTION FORMAT:

			P+1	P+2
7 6	5 4 3	2 1 0	7 0	7 0
1	c+4	2	LSP	MSP
			ADDRESS	

c: designates which flip-flop (condition) is to be tested.

NOTE

1. The condition of the selected flip-flop is unchanged by this instruction.
2. The stack is open-ended in operation. If it is overfilled, the deepest address will be lost.
3. Refer to Table 3-2 for list of flip-flop codes.

CONDITIONAL SUBROUTINE CALL

IF CONDITION FALSE: CFf_c

OP CODE: 1c2

TIMING: Variable if condition false, 24 usec. if condition true.

OPERATION: If (f_c=FALSE), P+3 → STACK, (P+1, P+2) → P.

DESCRIPTION: Examines the designated flip-flop. If reset, transfers the address of the next sequentially available instruction to the pushdown stack, and transfers control to the address of the two memory locations immediately following the Op Code. If the selected flip-flop is set, executes the next sequentially available instruction.

INSTRUCTION FORMAT:

			P+1	P+2
7 6	5 4 3	2 1 0	7 0	7 0
1	c	2	LSP	MSP
			ADDRESS	

c: designates which flip-flop (condition) is to be tested.

NOTE

1. The condition of the selected flip-flop is unchanged by this instruction.
2. The stack is open-ended in operation. If it is overfilled, the deepest address will be lost.
3. Refer to Table 3-2 for list of flip-flop codes.

SUBROUTINE RETURN: RETURN

OP CODE: 007 TIMING: Variable

OPERATION: (STACK) → P

DESCRIPTION: Transfer control to the address specified by the most recent entry in the Pushdown Stack. Deletes the most recent entry from the Stack.

INSTRUCTION FORMAT:

P

7 6	5 4 3	2 1 0
0	0	7

NOTE

1. The effect of attempting more "RETURN" than the Stack is capable of handling is undefined.

CONDITIONAL SUBROUTINE RETURN

IF CONDITION TRUE: RTf_c

OP CODE: 0(c+4)3

TIMING: Variable if condition true, 16 usec. if condition false.

OPERATION: If (f_c=TRUE), Stack → P. Otherwise P+1 → P

DESCRIPTION: Examines the designated flip-flop. If set, transfers control to the address specified by the most recent entry in the pushdown stack. Deletes the most recent entry in the stack. If the selected flip-flop is reset, executes the next sequentially available instruction.

INSTRUCTION FORMAT:

7 6	5 4 3	2 1 0
1	c+4	3

c: designates which flip-flop (condition) is to be tested.

NOTE

1. The condition of the selected flip-flop is unchanged by this instruction.
2. The effect of attempting more "RETURN" than the stack is capable of handling is undefined.
3. Refer to Table 3-2 for list of flip-flop codes.

CONDITIONAL SUBROUTINE RETURN

IF CONDITION FALSE: RFf_c

OP CODE: 0c3

TIMING: Variable if condition false, 16 usec. if condition true.

OPERATION: If (f_c= FALSE), Stack → P. Otherwise, P+1 → P

DESCRIPTION: Examines the designated flip-flop. If reset, transfers control to the address specified by the most recent entry in the stack. If the selected flip-flop is set, executes the next sequentially available instruction.

INSTRUCTION FORMAT:

7 6	5 4 3	2 1 0
1	c	3

c: designates which flip-flop (condition) is to be tested.

NOTE

1. The condition of the selected flip-flop is unchanged by this instruction.
2. The effect of attempting more "RETURN" than the stack is capable of handling is undefined.
3. Refer to Table 3-2 for list of flip-flop codes.

SHIFT RIGHT CIRCULAR: SRC

OP CODE: 012 TIMING: 16 usec.

OPERATION: $A_m \rightarrow A_{m-1}$, $A_o \rightarrow A_7$, $A_o \rightarrow C_f$

DESCRIPTION: Shifts the contents of the A register right in a circular fashion. Shifts the least significant bit into the most significant bit position. Upon completion of the operation, the Carry Flip-flop is equal to the most significant bit.

INSTRUCTION FORMAT:

P

7 6	5 4 3	2 1 0
0	1	2

NOTE

None of the flag flip-flops other than C_f is affected by this instruction.

SHIFT LEFT CIRCULAR: SLC

OP CODE: 002 TIMING: 16 usec.

OPERATION: $A_m \rightarrow A_{m+1}$, $A_7 \rightarrow A_o$, $A_7 \rightarrow C_f$

DESCRIPTION: Shifts the contents of the A register left in a circular fashion. Shifts the most significant bit into the least significant bit position. Upon completion of the operation, the Carry Flip-flop is equal to the least significant bit.

INSTRUCTION FORMAT:

P

7 6	5 4 3	2 1 0
0	0	2

NOTE

None of the flag flip-flops other than C_f is affected by this instruction.

NO OPERATION: NOP

OP CODE: 300 TIMING: 16 usec.

OPERATION: P+1 → P

DESCRIPTION: No instruction is executed.

INSTRUCTION FORMAT:

P

7 6	5 4 3	2 1 0
3	0	0

HALT: HALT

OP CODE: 000,001,377 TIMING: Execution Stops

OPERATION:

DESCRIPTION: The computer halts. When the START buttom on the console is depressed, operation resumes at P+1.

INSTRUCTION FORMAT:

P

7 6	5 4 3	2 1 0
0	0	0
0	0	1
3	3	7

INPUT: INPUT

OP CODE: 101 TIMING: 16 usec.

OPERATION: (I/O Bus) → A

DESCRIPTION: Transfers the contents of the I/O Bus to the A register.

INSTRUCTION FORMAT:

P

7 6	5 4 3	2 1 0
1	0	1

EXTERNAL COMMAND: EX (exp)

OP CODE: 121-177 depending on the specific command being executed.

TIMING: 16 usec.

OPERATION: Performs I/O control functions according to (exp)

DESCRIPTION: These instructions perform the functions necessary for control of the I/O system and external devices. Many of these functions are specifically related to operation of particular devices. The device oriented commands for the Keyboard, CRT Display, Cassette Tapes, and Communications Interface are explained in the sections covering these devices.

INSTRUCTION FORMAT:

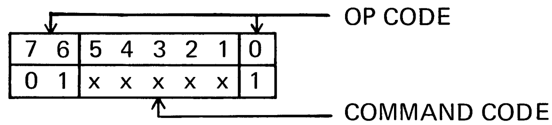


Table 3-3 is a list of External Commands used. For a detailed discussion of their use, reference should be made to Part 4 (Input/Output Operations) and to descriptions of the separate external devices.

TABLE 3-3

EXTERNAL COMMANDS

EX (exp)

COMMAND NUMBER	(exp)	OCTAL CODE	COMMAND	DESCRIPTION	DEVICE ADDRESS
1	ADR	121	Address	Selects device specified by A-register	ALL
2	STATUS	123	Sense Status	Connects selected device status to input lines	
3	DATA	125	Sense Data	Connects selected device data to input lines	
4	WRITE	127	Write Strobe	Signals selected device that output data word is on output lines	
5	COM1	131	Command 1	Outputs a control function to selected device	
6	COM2	133	Command 2	Outputs a control function to selected device	
7	COM3	135	Command 3	Outputs a control function to selected device	
8	COM4	137	Command 4	Outputs a control function to selected device	ALL
9	––	141	(Unassigned)	––	––
10	––	143	(Unassigned)	––	––
11	––	145	(Unassigned)	––	––
12	––	147	(Unassigned)	––	––

TABLE 3-3

EXTERNAL COMMANDS

EX (exp)

(Continued)

COMMAND NUMBER	(exp)	OCTAL CODE	COMMAND	DESCRIPTION	DEVICE ADDRESS
13	BEEP	151	Beep	Activates tone producing mechanism	341
14	CLICK	153	Click	Activates audible click producing mechanism	341
15	DECK1	155	Select Deck 1	Connects deck 1 to I/O bus	360
16	DECK2	157	Select Deck 2	Connects deck 2 to I/O bus	|
17	RBK	161	Read Block	Enables read circuitry and sets tape in forward motion	|
18	WBK	163	Write Block	Enables write circuitry and sets tape in forward motion	360
19	——	165	(Unassigned)	——	——
20	BSP	167	Backspace One Block	Backs up the selected tape one record	360
21	SF	171	Slew Forward	Sets selected tape deck in forward motion	|
22	SB	173	Slew Backward	Sets selected tape deck in backward motion	|
23	REWIND	175	Rewind	Rewinds the selected deck to beginning of tape	|
24	TSTOP	177	Stop Tape	Halts motion of the selected tape deck	360

TABLE 3-4

INSTRUCTION REPERTOIRE

OP CODE	MNEMONIC	OP CODE	MNEMONIC	OP CODE	MNEMONIC
000	HALT	050		120	JFS
001	HALT	051		121	EX ADR
002	SLC	052		122	CFS
003	RFC	053	RTZ	123	EX STATUS
004	AD	054	XR	124	
005		055		125	EX DATA
006	LA	056	LH	126	
007	RETURN	057		127	EX WRITE
010		060		130	JFP
011		061		131	EX COM1
012	SRC	062		132	CFP
013	RFZ	063	RTS	133	EX COM2
014	AC	064	OR	134	
015		065		135	EX COM3
016	LB	066	LL	136	
017		067		137	EX COM4
020		070		140	JTC
021		071		141	
022		072		142	CTC
023	RFS	073	RTP	143	
024	SU	074	CP	144	
025		075		145	
026	LC	076		146	
027		077		147	
030		100	JFC	150	JTZ
031		101	INPUT	151	EX BEEP
032		102	CFC	152	CTZ
033	RFP	103		153	EX CLICK
034	SB	104	JMP	154	
035		105		155	EX DECK1
036	LD	106	CALL	156	
037		107		157	EX DECK2
040		110	JFZ	160	JTS
041		111		161	EX RBK
042		112	CFZ	162	CTS
043	RTC	113		163	EX WBK
044	ND	114		164	
045		115		165	
046	LE	116		166	
047		117		167	EX BSP

TABLE 3-4

INSTRUCTION REPERTOIRE

(Continued)

OP CODE	MNEMONIC	OP CODE	MNEMONIC	OP CODE	MNEMONIC
170	JTP	240	NDA	310	LBA
171	EX SF	241	NDB	311	
172	CTP	242	NDC	312	LBC
173	EX SB	243	NDD	313	LBD
174		244	NDE	314	LBE
175	EX REWND	245	NDH	315	LBH
176		246	NDL	316	LBL
177	EX TSTOP	247	NDM	317	LBM
200	ADA	250	XRA	320	LCA
201	ADB	251	XRB	321	LCB
202	ADC	252	XRC	322	
203	ADD	253	XRD	323	LCD
204	ADE	254	XRE	324	LCE
205	ADH	255	XRH	325	LCH
206	ADL	256	XRL	326	LCL
207	ADM	257	XRM	327	LCM
210	ACA	260	ORA	330	LDA
211	ACB	261	ORB	331	LDB
212	ACC	262	ORC	332	LDC
213	ACD	263	ORD	333	
214	ACE	264	ORE	334	LDE
215	ACH	265	ORH	335	LDH
216	ACL	266	ORL	336	LDL
217	ACM	267	ORM	337	LDM
220	SUA	270	CPA	340	LEA
221	SUB	271	CPB	341	LEB
222	SUC	272	CPC	342	LEC
223	SUD	273	CPD	343	LED
224	SUE	274	CPE	344	
225	SUH	275	CPH	345	LEH
226	SUL	276	CPL	346	LEL
227	SUM	277	CPM	347	LEM
230	SBA	300	NOP	350	LHA
231	SBB	301	LAB	351	LHB
232	SBC	302	LAC	352	LHC
233	SBD	303	LAD	353	LHD
234	SBE	304	LAE	354	LHE
235	SBH	305	LAH	355	
236	SBL	306	LAL	356	LHL
237	SBM	307	LAM	357	LHM

TABLE 3-4

INSTRUCTION REPERTOIRE

(Continued)

OP CODE	MNEMONIC	OP CODE	MNEMONIC	OP CODE	MNEMONIC
360	LLA	370	LMA		
361	LLB	371	LMB		
362	LLC	372	LMC		
363	LLD	373	LMD		
364	LLE	374	LME		
365	LLH	375	LMH		
366		376	LML		
367	LLM	377	HALT		

NOTE

OP Codes shown without Mnemonics are undefined.

PART 4

INPUT/OUTPUT OPERATIONS

4.1 GENERAL

The versatile input/output capability of the Datapoint 2200 permits it to communicate with external devices (such as the 2200 communications adaptor) through a parallel I/O system. The keyboard, c.r.t. and tape decks that are an integral part of the Model 2200 perform all operations over the same I/O system as external devices.

4.2 INPUT/OUTPUT INSTRUCTIONS

Two types of instructions provide for I/O operations. One is the INPUT command (see section 3) which, upon execution, transfers whatever is on the input bus to the A-register. The second is the EXTERNAL command which is sub-divided into 24 separate command operations (8 of which are available to devices physically external to the Model 2200). Each EXTERNAL command produces a strobe pulse which may be used for control external to the processor. The actual control functions assigned to each external command are listed in Table 3-3.

4.3 INPUT/OUTPUT CABLE

The parallel I/O cable carries data, input strobe, external commands, and power between the 2200 processor and external devices connected to it. A complete I/O system is structured by connecting external devices in partyline fashion as shown in Figure 2-1. The I/O cable contains 8 input data lines, 8 output data lines, 1 input strobe line, 8 (of the 24) external command lines, 1 clock line, and 7 power and ground lines.

4.4 I/O DATA LINES

The data lines are broken into two groups. 8 lines are used for output and 8 lines are used for input.

The data output lines are connected (at all times) to the A-register in the processor and are used to perform three basic functions:

a. To transfer an address to select an external device (including the keyboard, c.r.t. and tape decks);

b. To transfer commands to an addressed device; and

c. To transfer data to an addressed device.

The data input lines are strobed into the A-register upon execution of the INPUT instruction and used to perform two basic functions.

a. To transfer status information from an addressed external device; and

b. To transfer data from an addressed external device.

As shown in Figure 4-1, input data or status from the data input lines is processed through input receivers and gated into the A-register. Once in the A-register data can be manipulated or stored as desired. Addresses, commands, or data that is to be transferred to an external device must first be loaded into the A-register. From the A-register it is transmitted through output devices onto the data output lines. The A-register, then, is used as a buffer register between the 2200 processor and external devices for all input and output data transfers.

4.5 INPUT STROBE

The INPUT STROBE carries a signal (8 usec. pulse) from the processor to the external device to indicate that whatever data is on the data input lines has been sampled and transferred into the A-register. The trailing edge of the pulse may be used by an external device to remove data from the data input line or to clear a status bit. The INPUT strobe is generated upon execution of the INPUT instruction.

4.6 EXTERNAL COMMAND STROBES

The eight EXTERNAL commands used by devices physically external to the Model 2200 are given function assignments as follows:

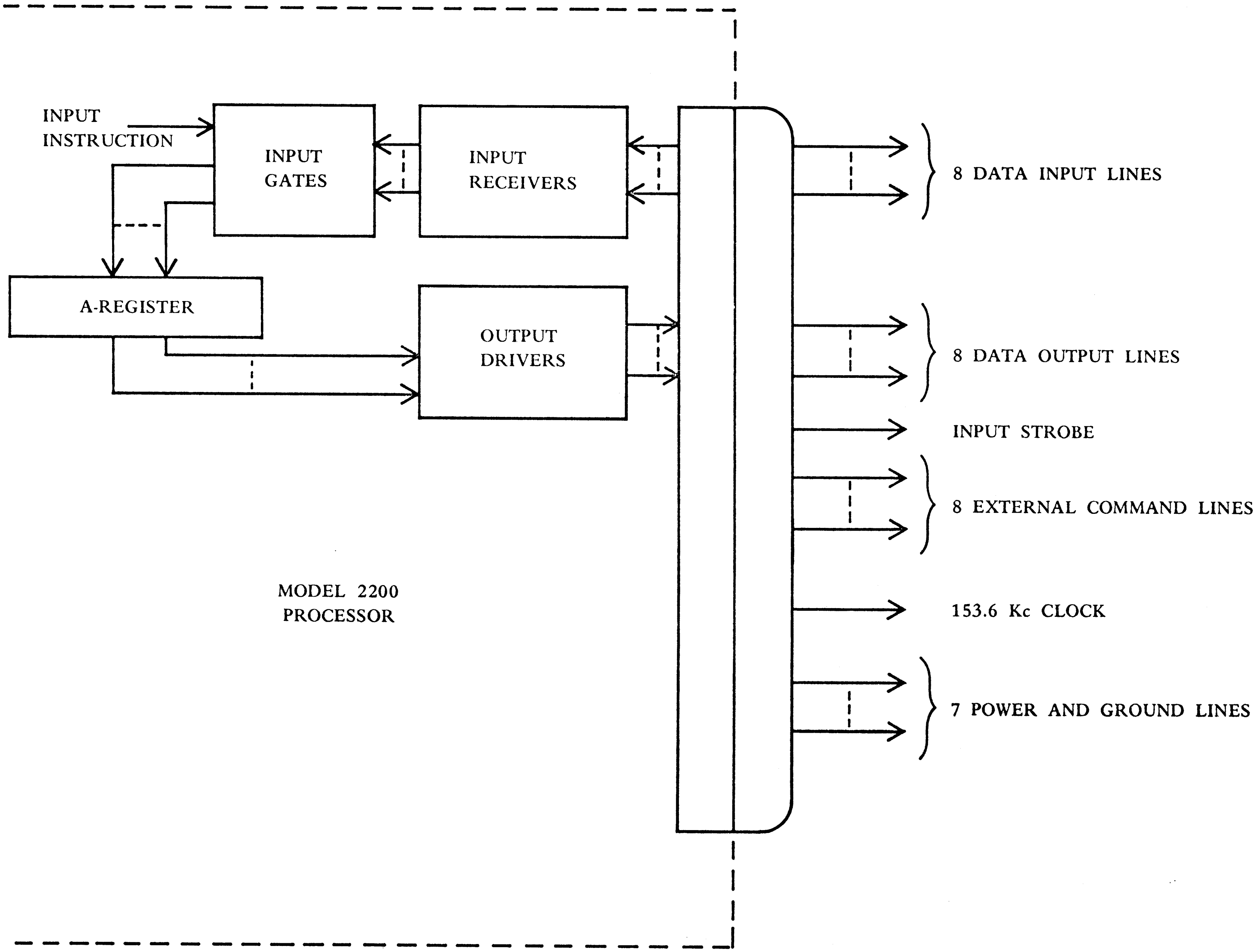


FIGURE 4-1

I/O SYSTEM, FUNCTIONAL DIAGRAM

TABLE 4-1

EXTERNAL COMMANDS

EX (exp)

COMMAND NUMBER	(exp)	OCTAL CODE	COMMAND	DESCRIPTION
1	ADR	121	Address	Selects device specified by A-register
2	STATUS	123	Sense Status	Connects selected device status lines to data input bus
3	DATA	125	Sense Data	Connects selected device data lines to data input bus
4	WRITE	127	Write Strobe	Signals selected device that output data is on data output lines
5	COM1	131	Command 1	Signals selected device that a control word is on data output lines
6	COM2	133	Command 2	Signals selected device that a control word is on data output lines
7	COM3	135	Command 3	Signals selected device that a control word is on data output lines
8	COM4	137	Command 4	Signals selected device that a control word is on data output lines

Execution of an EXTERNAL instruction provides a pulse 8 microseconds long. No functions are performed within the 2200 processor during execution of an EXTERNAL instruction. The interpretation of each of the EXTERNAL instructions is as follows:

a. Address. The address command (EX ADR) is a signal from the processor to all external devices to indicate that the information on the data output bus is to be interpreted as an external device address. Whenever an address command is executed all external devices should be disconnected from the I/O system except the device whose address appears in the A-register. (See paragraph 4.10 for discussion of address assignments).

b. Sense Status. The sense status (EX STATUS) command is a signal from the processor to the selected external device to place status information on the data input lines. (Note: External devices should be configured such that status is connected to the data input line whenever the device is first addressed. It is only necessary to use the EX STATUS instruction when it is desired to sense status after an EX DATA instruction has been used and a new address sequence has not been executed).

c. Sense Data. The sense data (EX DATA) command is a signal from the processor to the selected external device to place its data on the data input lines.

d. Write Strobe. The write strobe (EX WRITE) command is a signal from the processor that data is present on the data output lines for the selected external device.

e. Command 1 through Command 4. Command 1 through Command 4 (EX COM1, etc.) have meaning appropriate to the device selected. Reference should be made to a description of each device for specific function assignments.

4.7 CLOCK LINE

The clock line is crystal controlled 153.6 kilohertz squarewave that is available to external devices for timing purposes.

4.8 I/O BUS ELECTRICAL SPECIFICATIONS

All signals in the I/O System operate with a voltage swing of zero to +5 volts. Line drivers have a source impedance of approximately 470 ohms and line receivers have an input impedance in excess of 18,000 ohms and a decision threshold of +1.7 volts. Figure 4-2 illustrates a typical output line circuit.

All logic levels are True (logical 1) for zero (less than +1.7) volts and False (logical 0) for +5 (greater than 1.7) volts.

4.9 DATA TRANSFER OPERATION

a. Data Output. Figure 4-3 illustrates the sequence of events that occur when data is transferred from the

2200 processor to an external device. A typical program sequence to execute a transfer is as follows:

WDATA	LA 0322	Load device address into A-register
	EX ADR	
	INPUT	Load device status into A-register
	SRC	Shift desired status bit into C flip-flop
	JFC EXIT	Exit if device not ready
	LAM	Load A-register with DATA
	EX WRITE	Write Data to device

Once a device is addressed it remains addressed until another device is addressed so that succeeding commands may be transmitted to a device without re-addressing the device. Transmitting a command to a device would follow a program sequence similar to a data transfer except that EX COM_n would replace EX WRITE.

b. Data Input. Figure 4-4 illustrates a sequence of events that occur when data is transferred from an external device to the 2200 processor. A typical program sequence is as follows:

RDATA	LA 0322	Load A-register with device address
	EX ADR	
	INPUT	Load device status into A-register
	SRC	Shift status bit into C flip-flop
	SRC	
	JFC EXIT	Exit if device not ready
	EX DATA	Place data on input lines
	INPUT	Load A-register with data

4.10 DEVICE ADDRESS NUMBERING

Address assignments in the I/O system provides for up to 16 devices external to the 2200 processor. The address word is formulated such that the low-order four bits form the binary value for the address and the high-order four bits form the logical complement of the low order bits. For example device number 6 would have an address word as follows:

1	0	0	1	0	1	1	0	A-Register

LOGICAL COMPLEMENT BINARY 6 — BINARY 6

OCTAL 226

This addressing system permits any device to be coded for its particular address with only a four-input gate strapped to those output lines that are set to one during the address command. In addition, all devices can be cleared by setting the A-register to all zeros and executing an EX ADR instruction.

Device addresses used in the Model 2200 are given in the following table:

TABLE 4-2

DEVICE ADDRESS ASSIGNMENTS

DEVICE	NUMBER	BINARY	OCTAL
Cassette Tape Decks	0	11110000	360
CRT/Keyboard	1	11100001	341
Communications Adaptor	2	11010010	322
2200P Printer	3	11000011	303
2200T Tape Transport	4	10110100	264
Unassigned	5	10100101	245
"	6	10010110	226
"	7	10000111	207
"	8	01111000	170
"	9	01101001	151
"	10	01011010	132
"	11	01001011	113
"	12	00111100	074
"	13	00101101	055
"	14	00011110	036
"	15	00001111	017

4.11 I/O POWER AND GROUND LINES

The Model 2200 provides several power supply voltages for use by external devices. Table 4-3 below lists the characteristics of each power and ground line.

4.12 I/O SYSTEM CONNECTOR

Connection to the I/O system is made through an Amphenol 17-20500-1 connector. The mating I/O cable should have a 50-pin Amphenol 17-10500-1 connector.

Table 5-5 lists the pin assignments.

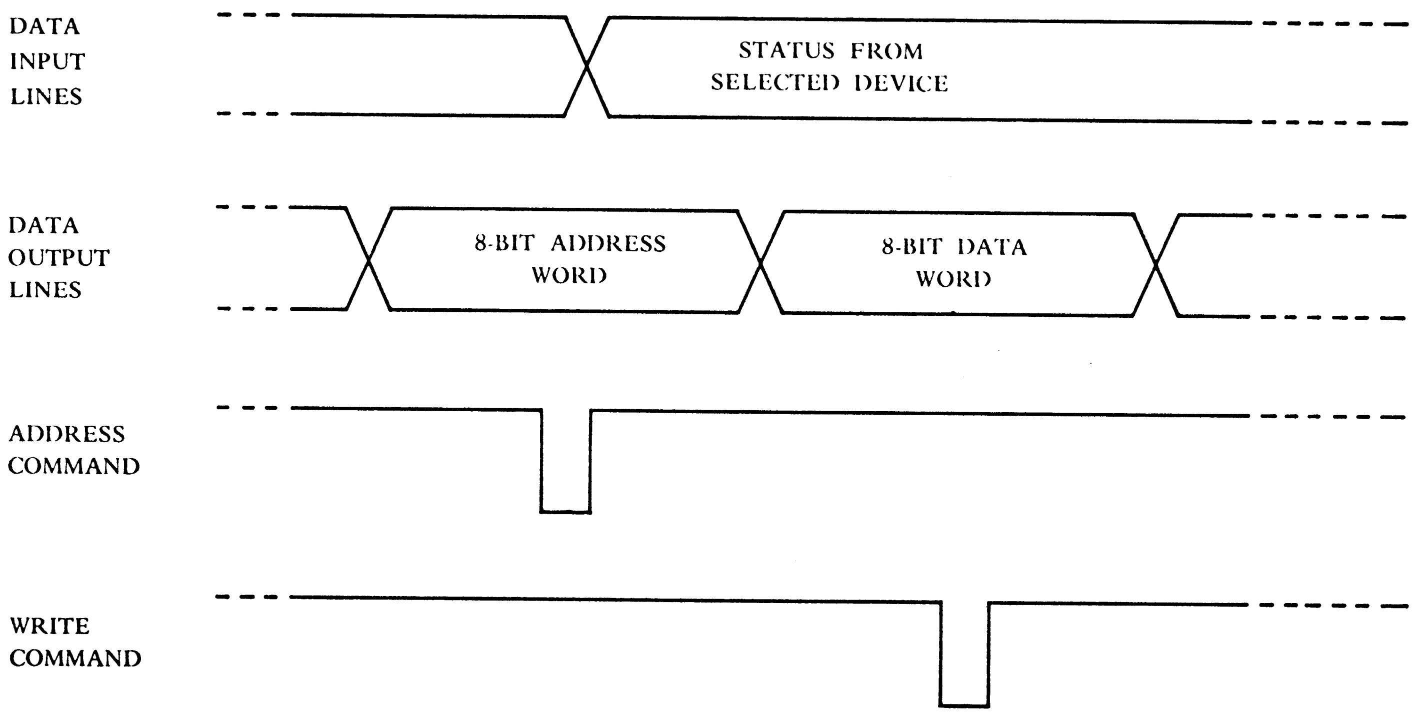


FIGURE 4-3

TYPICAL DATA OUTPUT
SEQUENCE

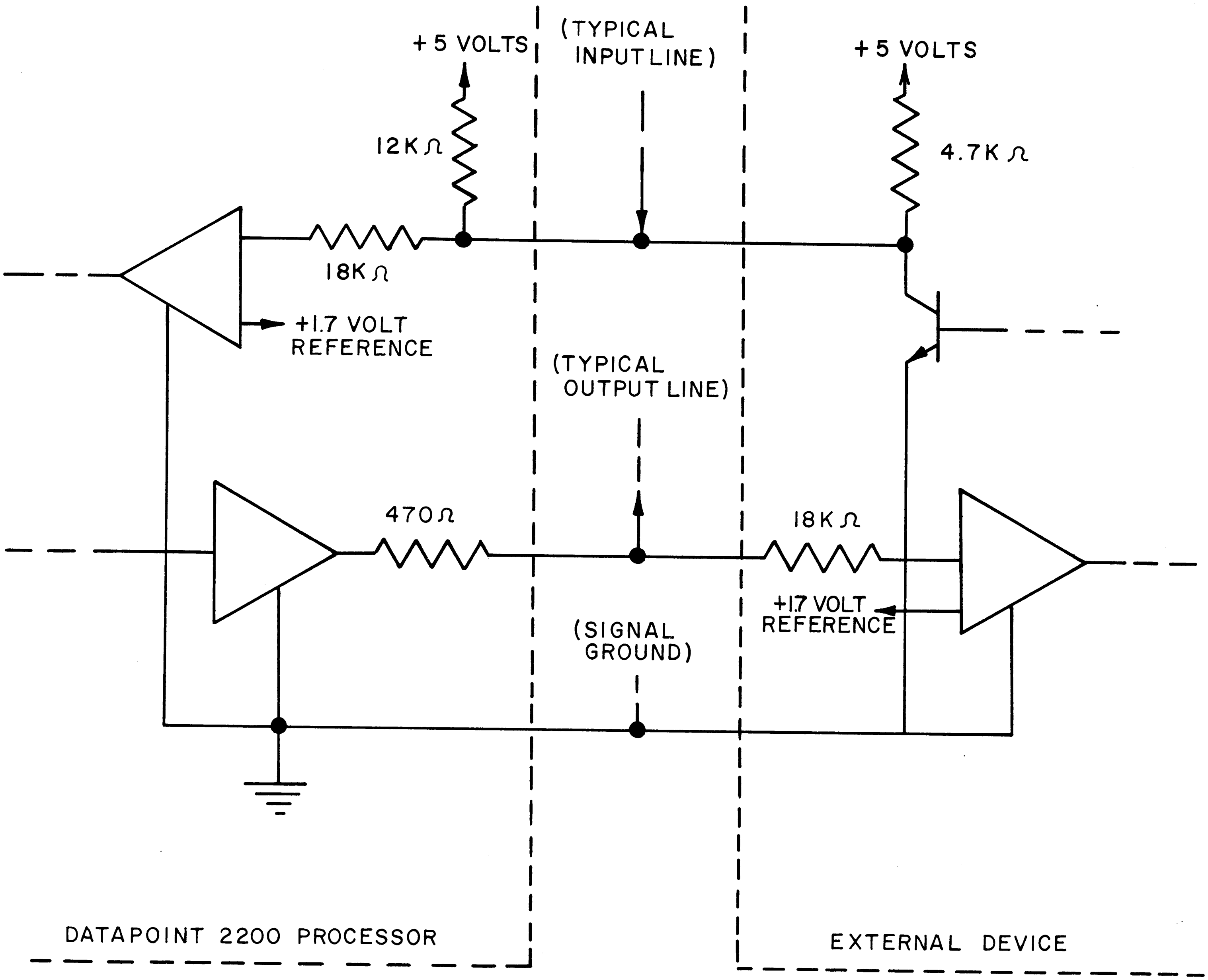


FIGURE 4-2

I/O CABLE, ELECTRICAL CHARACTERISTICS

TABLE 4-3

I/O POWER AND GROUND LINES

VOLTAGE	MAX. CURRENT	REGULATION
-12 Volts	0.5 amps	±10%
- 5 Volts	0.1 amps	±5%
+ 5 Volts	3.4 amps	±5%
+12 Volts	0.5 amps	±10%
+24 Volts	0.1 amps	±5%
Power Ground	—	—
Signal Ground	—	—

TABLE 4-4

I/O CONNECTOR PIN ASSIGNMENTS

ASSIGNMENT		PIN NUMBER
Data output	0	44
	1	45
(A Bus Outputs)	2	46
	3	29
	4	30
	5	31
	6	32
	7	33
Data Input	0	1
	1	2
(A Bus Inputs)	2	3
	3	4
	4	5
	5	6
	6	7
	7	18
Input Strobe (Read)		12
Address Command		15
Sense Status Command		13
Sense Data Command		14
Write Command		19
Command 1		20
Command 2		21
Command 3		22
Command 4		23
153.6 KHz Clock		39
-12v		24
-5v		27
+5v		8, 9, 10, 11
+12v		25
+24v		26
Ground (Power & Signal)		40, 41, 42, 43

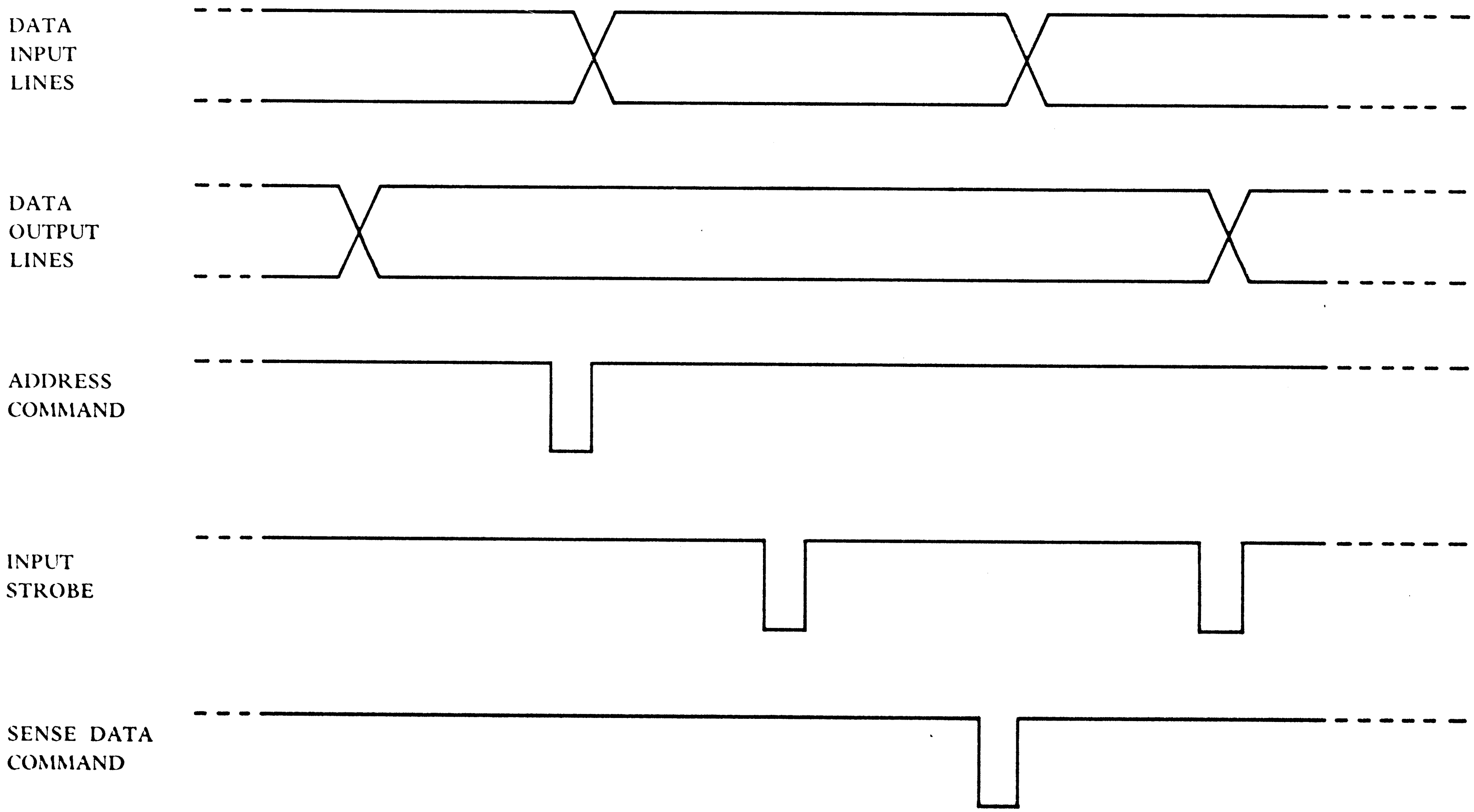


FIGURE 4-4

TYPICAL DATA INPUT
SEQUENCE

PART 5

KEYBOARD

5.1 GENERAL DESCRIPTION

The keyboard on the Datapoint 2200 performs the functions of data entry and processor control. The keys are divided into three sections, each of which has it own function.

Section 1 consists of 41 standard alphabetic, numeric and special character keys found in the ASCII character set. Figure 5-1 illustrates the keyboard layout.

Section 2 consists of an 11 key matrix which is identical to a standard adding machine keyboard with the addition of a decimal point (period). The keys in this section are duplicates of certain keys found in Section 1 and are provided to facilitate entry of large amounts of numeric data.

The keys in Section 3 are special function keys which exert control over the processor. Their names and associated functions are as follows:

RUN — Momentary contact switch, which when depressed, causes the processor to begin execution of the instruction located at the address in memory currently addressed by the program counter.

STOP — Momentary contact switch which, when depressed, causes instruction execution to halt at the completion of the current instruction. Care should be taken when using this switch, because any tape operation which may be in progress will be aborted.

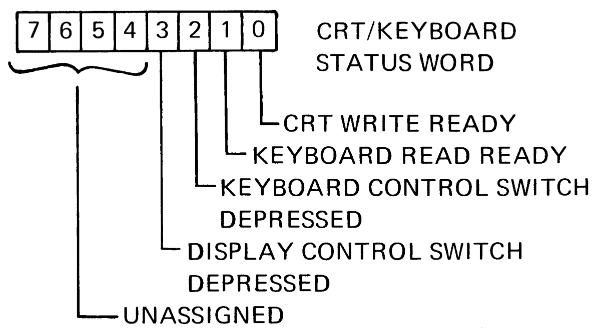
KEYBOARD — Momentary contact switch which sets a status bit that may be tested at any time by the processor.

DISPLAY — Momentary contact switch with a function similar to that of KEYBOARD switch. Either one or both of these switches may be depressed.

RESTART — Momentary contact switch which causes the processor to halt, rewind the system or program tape mounted on Deck 1, load and execute the first record found on tape.

5.2 OPERATION

The keyboard is addressed by the processor by loading the A-register with 341_8 and executing an EX ADR command. (The crt display also uses this address. Data transfers to the processor are from the keyboard and transfers from the processor are to the display). Following the address sequence the c.r.t./keyboard status word can be loaded into the A-register by executing an INPUT instruction. Bit 1 of the A-register may be tested by the program to determine if a character is ready for transfer from the keyboard. Bits 2 and 3 will indicate if either the KEYBOARD or DISPLAY control switch is pressed.



The External Commands associated with the operation of the keyboard are as follows:

a. EX BEEP. This command produces a 1500 Hertz tone for a duration of about 100 msec. The tone could be used as an error or ready signal to the keyboard operator.

b. EX CLICK. This command produces an audible click which could be used to acknowledge receipt of a valid character when a key is depressed.

c. EX COM1 (Command 1). Presents a control word contained in the A-register to the keyboard. Bit 5 of the control word controls the KEYBOARD switch light and bit 6 controls the DISPLAY switch light as follows:

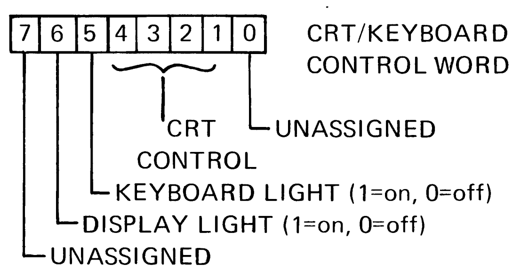


TABLE 5-1

KEYBOARD CODING (ASCII)

A-101	a -141	0-060	: -072
B-102	b -142	1-061	; -073
C-103	c -143	2-062	< -074
D-104	d -144	3-063	= -075
E-105	e -145	4-064	> -076
F-106	f -146	5-065	? -077
G-107	g -147	6-066	[-133
H-110	h -150	7-067	~ -176
I -111	i -151	8-070	] -135
J -112	j -152	9-071	Λ -136
		Space-040	— -137
K-113	k -153		
L-114	l -154	!-041	@ -100
M-115	m-155	"-042	{ -173
N-116	n -156	#-043	\ -134
O-117	o -157	$-044	' -140
P-120	p -160	%-045	| -174
Q-121	q -161	&-046	} -175
R-122	r -162	'-047	Enter-015
S-123	s -163	(-050	Cancel-030
T-124	t -164	)-051	Backspace-010
U-125	u -165	*-052	Rubout (R.O.)-177
V-126	v -166	+-053	
W-127	w-167	,-054	
X-130	x -170	--055	
Y-131	y -171	.-056	
Z-132	z -172	/-057	

PART 6

CRT DISPLAY

6.1 GENERAL DESCRIPTION

The display unit on the Datapoint 2200 consists of a CRT capable of displaying 12 lines of 80 characters each, a character generator, 960 cells of refresh memory (refresh rate 60 Hz), and a group of registers utilized to position the cursor. Maximum character transfer rate to the CRT is 60 characters per second.

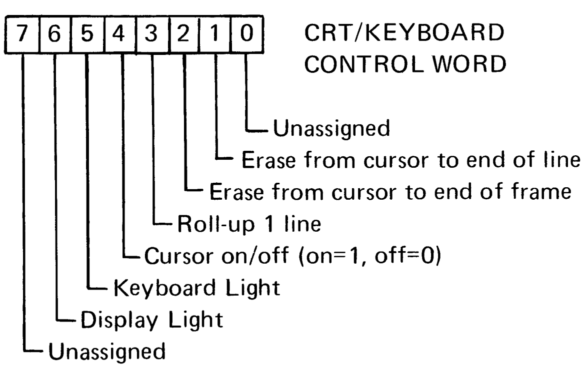
The character set utilized by the CRT display consists of the full ASCII set with both upper and lower case alphabetics and all numeric and special characters.

6.2 OPERATION

The CRT is addressed and status tested in the same manner as the keyboard (see paragraph 5.2). Bit 0 of the status word indicates that the CRT is ready to accept data or commands. Characters are transferred to the screen by loading the A-register with the character to be displayed and executing an EX WRITE. The character will be displayed at the current cursor location.

Control of the CRT is accomplished through the use of the three external commands - Command 1, Command 2, and Command 3. The functions performed by these commands are as follows:

a. EX COM1 (Command 1) Transfers a control word contained in the A-register to the CRT. The applicable bit assignments and their functions are as follows:



The erase functions permit selective erasures on the screen by limiting erasures to those character positions following the current cursor position to the end of the line (or page).

The roll-up function causes all displayed characters (not the cursor) to move up one line. The top line on the screen is lost.

The cursor image may be turned on or off through the control word. The cursor position is the same in either case. The cursor image is automatically turned off whenever the processor is in the HALT state.

b. EX COM2 (Command 2) Positions the cursor to the horizontal character slot designated by the contents of the A-register. Character position $0\text{-}79_{10}(0\text{-}117_8)$ are valid.

c. EX COM3 (Command 3) Positions the cursor to the line designated by the contents of the A-register. Line number $0\text{-}11_{10}$ $(0\text{-}13_8)$ are valid.

In order to write a new character, the cursor must occupy that character's position on the screen. After the character has been written, the cursor should then be moved to the next horizontal (or vertical) position desired. The CRT Write Ready status bit must be true before positioning the cursor or displaying a character.

Both the CRT and keyboard utilize the standard ASCII character set. (See Table 5-1). Any invalid character code will appear as a blank space on the CRT screen.

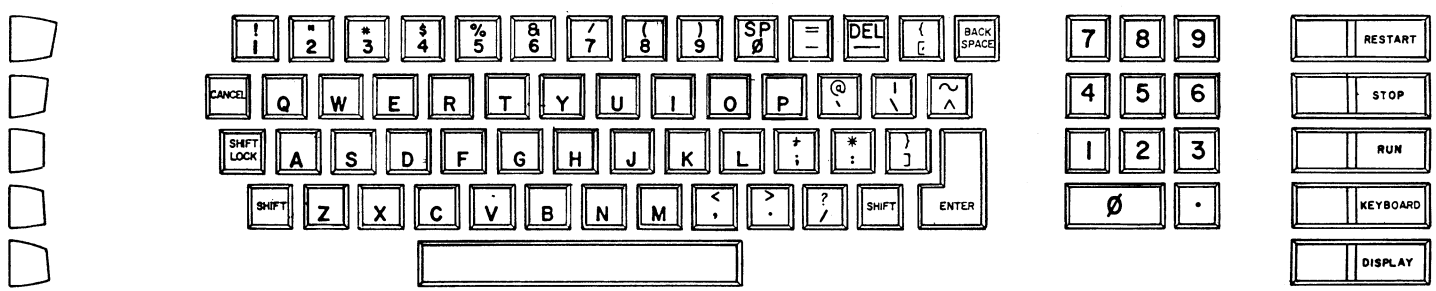


FIGURE 6-1
KEYBOARD LAYOUT

PART 7

CASSETTE TAPES

7.1 GENERAL DESCRIPTION

The Datapoint 2200 contains two cassette tape recording devices for storage of programs and data. Since the hardware RESTART (section 5.1) uses the rear deck (number one), programs will typically be on it while data areas will be the front deck (number two). However, once the machine is initially loaded, either deck may be used for both purposes.

Data on the Tape is organized by record (of any length). Records are written and read at 350 eight-bit characters per second with a tape speed of approximately 7.5 inches per second. See Table 7.1 for a list of the physical specifications.

7.2 OPERATION

Data is recorded or read in bit serial fashion on one track. Each eight bit character is framed by three sync bits on either side of the character:

-	0 1 0	x x x x x x x x	0 1 0	x x x x x x x x	0 1 0	-
	Sync Code	Character 1	Sync Code	Character 2	Sync Code	

The appearance of the correct sync code indicates that the character is valid. Any other sync code causes special action to be taken on data reads. Note that the sync codes are valid for tape motion in either direction so the tape may be read backwards although in the reverse direction the data bits will apear reverse d (bit 0 will be bit 7, 1 will be 6 etc.)

A record is a group of successive valid characters. An inter-record gap is indicated by the failure of the sync code to be zero one zero and all mark code. (ones):

-	0 1 0	x x x x x x x x	0 1 0	x x x x x x x x	1 1 1	x x x	-
	Sync Code	Valid Character	Sync Code	Invalid Character	Sync Code	Inter Record Gap	

Only valid characters will be presented as data from the tape unit.

7.3 STATUS

The cassette tape unit is addressed by the processor by loading the A-register with 360_8 and executing the EX ADR instruction. Following this sequence, the tape unit status can be loaded into the A-register by executing an INPUT instruction. The bit assignments are as follows:

```
|7|6|5|4|3|2|1|0|   TAPE STATUS WORD
 | | | | | | | |
 | | | | | | | └ Deck Ready
 | | | | | | └ End of Tape
 | | | | | └ Read Ready
 | | | | └ Write Ready
 | | | └ Inter-Record Gap
 | | └ Unassigned
 | └ Cassette in Place
 └ Unassigned
```

DECK READY	Deck ready will be set whenever the tape unit is ready to accept another command. (Only the TSTOP command should be issued if this bit is false). The tape will be stopped, a cassette in the selected deck and not wound to the clear leader at either end, and the head engaged when this bit is true. This bit should be checked after selecting a deck.
END OF TAPE	End of Tape indicates that the cassette has run onto leader (in either direction).
READ READY	Read Ready indicates that the selected deck has read another character.
WRITE READY	Write Ready indicates that the selected deck is ready to write another character.
INTER-RECORD GAP	Inter-Record Gap indicates the selected deck has come across an inter-record gap (invalid sync code).
CASSETTE IN PLACE	Cassette in Place indicates that a cassette is physically in place in the selected deck.

7.4 CONTROL

When the cassette tape unit is addressed the following instructions will control the action of the tape:

a. EX TSTOP causes any motion of either deck to be stopped, any read or write operations to be terminated. When everything has settled, the ready status bit will come true and operations may be resumed.

b. EX DECK1 causes deck one (rear) to be the currently selected deck. Before commanding a deck selection, care should be taken that the currently selected deck has completed all operations.

c. EX DECK2 causes deck two (front) to be the currently selected deck. Note the precaution in (b).

d. EX RBK causes the currently selected deck to be set in forward motion and, after 70 msec, for the read circuitry to be enabled. The read ready status bit will come true upon appearance on the tape of the first valid character. Upon appearance of an invalid sync code, the inter-record gap status bit comes true and tape motion is automatically stopped. Note that this will happen only after at least one valid character has been found. Once the read ready status bit comes true, the character must be taken within 2.8 milliseconds or it will be overwritten with the next one. The tape read hardware double-buffers incoming characters to allow the 2.8 msec character availability.

e. EX BSP is similar to EX RBK except that tape motion is in the reverse direction so the data bits will be reversed.

f. EX SF is similar to EX RBK except the tape is not stopped upon appearance of an inter-record gap, and if allowed to continue will start to read the next record on the tape. In this case, the read ready status bit will come true again after the first character of the next record is read. Only an EX TSTOP will stop the motion initiated by EX SF.

g. EX WBK causes the currently selected deck to be set in forward motion and for all status bits except the write ready to go false. A character must then be presented within 2.8 milliseconds (the first character will be accepted at once due to the buffering in the tape hardware and then there will be a pause while the tape comes up to speed), at which time the write ready will go false until the writing circuitry is ready to accept another character. An end of record is a signalled to the hardware by withholding a character for a period of time longer than 2.8 milliseconds specified above. When this is done, the write ready will go false, an inter-record gap will be written, the tape motion will cease, and the deck ready status bit will come true again.

h. EX REWIND causes the tape to be rewound to the beginning on the selected deck. Worst case rewind time is approximately 40 seconds.

i. PUNCH TABS, on the Cassette Cartridge are used for "write protect" and "automatic restart". The punch tab on the left (as you face the terminal) inhibits the ability to write on tape, when punched. When the tab on the right is punched, it causes an automatic restart whenever a halt or power-up occurs.

TABLE 7-1

TAPE UNIT PHYSICAL SPECIFICATIONS

Density	47 characters/inch
Speed	7.5 ips
Recording Rate	350 c.p.s.
Capacity	130,000 characters (typical)
Start/Stop Time (Inter-Record Gap)	280 msec.
Start/Stop Distance (Inter-Recrod Gap)	2 inches
Rewind Speed	90 ips
Rewind Time (max 300 ft.)	40 sec.
Character Transfer Time	2.8 msec.

PART 8

COMMUNICATIONS ADAPTOR

8.1 GENERAL DESCRIPTION

The 2200 Communications Adaptor is an external device, which when connected to the Datapoint 2200 Input/Output System permits asysnchronous serial data interchange to other remote systems or devices.

The Communications Adaptor consists of three basic parts:

a. The serial data transmitter and time base;

b. The serial data receiver and time base; and

c. The communications channel interface.

The communications channel interface may be one of four types:

a. An EIA RS-232 type interface;

b. An isolated high-level neutral or polar telegraph loop interface;

c. A modem compatible with the Bell System 103 type modems;

d. A modem compatibel with the Bell System 202 type modems.

8.2 OPERATION

The serial data transmitter and receiver are addressed at the same time (the address of the first used communications adaptor is 322_8 - see Table 4-2). Additional adaptors may be given previously unassigned addresses.

To set the bit rate desired for the transmitter time base two successive EX COM3 instructions are used to transfer two 8-bit masks from the A-register (See paragraph 8.6 for a discussion of time base mask words). For the receiver EX COM2 is used.

To set the character length for the transmitter and receiver an EX COM4 command is executed with a character length mask from the A-register (see paragraph 8.7 for a discussion of character length mask words).

The status of the communications adaptor is transmitted to the A-register with the following bit assignments:

Communications Adaptor Status Word

Bit	Meaning
0	Transmit Ready
1	Receive Ready
2	Break Received
3	*Clear to Send
4	*Carrier Present, Reverse Channel
5	*Ringing Present
6	*Carrier Present, Main Channel
7	*Dial tone present (originate mode) *Data coupler ready (answer mode)

*Used with data set options.

Communications Adaptor Status Bit Description

Bit 0, Transmit Ready

The "true" condition of this bit indicates that the serial transmitter is ready to accept a new character for transmission. Should another write command be issued to the Communications Adaptor while this bit is "false", i.e. transmitter NOT ready, the previous character will be written over.

Bit 1, Receiver Ready

The Receive Ready bit, in the true state, indicates the presence of a new received character. A read command to the Communications Adaptor returns this bit to the false state. If a read command is not issued before another new character is received, the new character will replace the existing character and the status will remain true.

Bit 2, Break Received

The Break Received status bit simply indicates that the received data is in the "space" or "zero" condition for longer than one character time.

Bit 3, Clear to Send

The true state of Clear to Send status indicates that the data set (internal or external) is prepared to accept data for transmission. This bit has meaning only when an internal or external data set is in used.

Bit 4, Carrier Present - Reverse Channel

This status bit has significance only when operating half-duplex with either an internal or external 202 type data set (modem). The true condition indicates that the reverse (supervisory) channel carrier is being received.

Bit 5, Ringing Present

The true condition of Ringing Present indicates that the ringing of an incoming call has been detected. This bit has significance only when used with an internal or external (with proper options) data set.

Bit 6, Carrier Present - Main Channel

The true condition of this status bit indicates that the main channel carrier is being received. This status bit has meaning only when used with an internal or external data set.

Bit 7, (1) Dial Tone Present (Originate Mode)
(2) Data Coupler Ready (Answer Mode)

(1) When originating a call, the true condition of this status bit indicates that a dial tone is present and dialing may proceed; during dialing, the status will become false. Following dialing, and a 2 to 5 second delay, this bit will return to the true condition indicating connection to the telecommunication network (but does not indicate the called number has answered).

(2) When answering a call, the true condition of this status bit indicates that the data coupler is connected to the telecommunications network.

8.3 DATA OUTPUT

After addressing the communications adaptor transmission of each character is accomplished in the following manner:

a. Input the status word and verify that status bit 0, Transmit Ready, is set to 1 indicating that the adaptor can accept another character.

b. Load the A-register with the byte to be transmitted.

c. Apply a write strobe (EX WRITE). Data present on the A-bus will be loaded into the data transmitter and data will be serially transmitted at the selected code length and bit rate.

8.4 DATA INPUT

After addressing the communications adaptor, reception of each character is accomplished in the following manner:

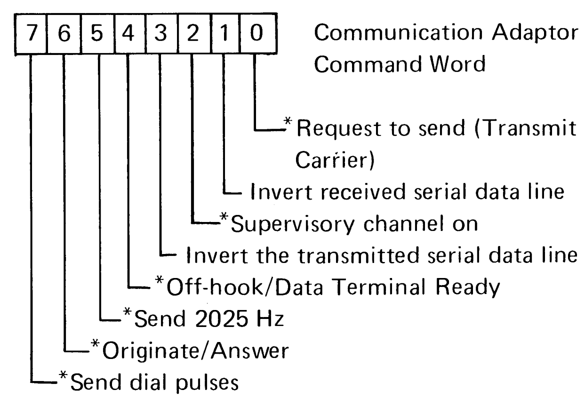
a. Input the status word and verify that status bit 1, Receiver Ready, is set to 1, indicating that a character has been received.

b. Execute an EX DATA instruction.

c. Execute an INPUT command, transferring the received character to the A-register.

8.5 COMMAND WORD

Control of the communications adaptor is accomplished through the use of a command word. The command word is transmitted to the adaptor by executing EX COM1.



*Used with data set options

Communication Adaptor Command Word Description

Bit 0 - Request to Send

This command bit controls the transmit carrier of an internal or external data set. A "one" in this position turns on the transmit carrier and indicates to the data set that it must prepare for data transmission.

Bit 1 - Invert Received Serial Data Line

A "one" in this position permits data to be received normally when the received serial data line is inverted.

Bit 2 - Supervisory Channel On

This command is used only with a 202 type modem in half-duplex operation. A "one" in this command indicates to the modem that the supervisory (or reverse) channel will be operative, transmit or receive.

Bit 3 - Invert Transmitted Serial Data

A "one" in this command inverts the transmitted serial data.

Bit 4 - Off-Hook

A "one" must be placed in this bit position any time a telecommunication call is to be originated or answered. This command allows connection to be made to the telecommunication network with an internal modem and a Bell System Data Access Arrangement. When using an external modem, this command provides "Data Terminal Ready" to the external modem, i.e., the system is prepared for on-line communications. This command is used only for the cases described above.

Bit 5 - Send 2025

This command is used only with an internal 202 type modem, half-duplex operation and "answer" mode. The only use of this command is described as follows:

1.) following receipt of Ringing Present, Status Bit 5, the Off Hook Command, Command Bit 4, is set to a "one".

2.) Next, Status Bit 7, Data Coupler Ready must become "true".

3.) Send 2025 command must now be set to a "one" only for a period of 1/2 second to 3 seconds to inform the calling data set of our response.

Bit 6 - Originate

This command is used only with internal data sets (modems). A "one" in this command instructs the modem that the system will originate a telecommunication call, A "zero" tells the modem the system is prepared to answer a telecommunication call.

Bit 7 - Send Dial Pulses

This command is used only with internal data sets (modems) and is set to "one" only when dialing. Its use is described as follows:

1.) Off-Hook Command (Bit 4) is set to "one".

2.) Status Bit 7 - Dial Tone Present becomes "true".

3.) Bit 7, Bit 4 and Bit 3 (invert xmit), are now set to "one".

4.) When the last dial pulse is completely transmitted, Bit 7 and Bit 3 must be returned to "zero".

8.6 TIME BASE MASK WORDS

Both time base generators are programmed for their respective bit rates by the processor. Each time base is independently controlled to allow transmission and reception at different rates.

After addressing the communication interface, two eight-bit mask words are loaded into the time base registers to synthesize the selected bit rates. As each respective byte is presented, a corresponding EX COM2 instruction must be executed to load the receive time base and an EX COM3 instruction to load the transmit time base.

These two bytes are combined to form a 16 bit word which is placed in a holding register. A counter is then set to the value in the holding register. This counter is incremented at the rate of 153,600 Hz. Each time the counter overflows, i.e., goes from all one to all zeroes, a pulse is generated and the counter is reset to the value in the holding register. The time between pulses represents 1/2 clock period or 1/2 bit time. Given a bit rate (bps), the following formula can be used to determine the number N to be entered into the holding register:

$$N = 65{,}536 - \left(\frac{76{,}800}{\text{bps}}\right)$$

This number N may then be converted to a 16 bit binary number and separated into the two 8-bit mask words.

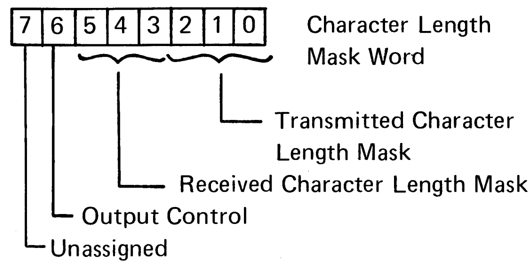
The octal codes for some of the more frequently used rates are listed below:

BIT RATE	1ST MASK WORD	2ND MASK WORD
1) 100*	375	000
2) 110	375	106
3) 220	376	243
4) 440	377	121
5) 150	376	000
6) 300	377	000
7) 600	377	200
8) 1200	377	300
9) 2400	377	340
10) 4800	377	360
11) 9600	377	370

*(Dialing)

8.7 CHARACTER LENGTHS

Character lengths for the transmit and receive sections and its output control bit are determined by a character length mask word which is transmitted to the communications adaptor with an EX COM4 instruction.



The functions of the mask bits are given in the following tables:

TABLE 8-1

TRANSMITTED CHARACTER LENGTH MASK BITS

MASK BIT POSITION 210	START UNITS	INFORMATION UNITS	STOP UNITS	CODE BIT POSITIONS 76543210
000	1	8	1	87654321
001	1	8	2	87654321
010	1	7	1	7654321
011	1	6	1	654321
100	1	5	1	54321
101	–	–	–	––
110	–	–	–	––
111	–	–	–	––

When codes having 5, 6, or 7 information units are to be transmitted, the remaining high-order bits in the character byte must be coded to "1".

When a two-unit stop pulse is required for characters having 5, 6, or 7 information bits, the next larger character length is used; the remaining high-order bits (all coded 1) form the stop pulses.

When received characters contain 5, 6 or 7 information bits, the remaining low-order bits (as shown above) must be disregarded.

One additional command bit, Bit 6 (output control) of the Character Length Mask Word, is used to control the EIA RS-232 Transmitted Data and the High-Level Keyer Transmitted Data. A "one" in this command bit enables serial data to be transmitted only to the EIA RS-232 output or to

TABLE 8-2

RECEIVED CHARACTER LENGTH MASK BITS

MASK BIT POSITION 543	START UNITS	INFORMATION UNITS	STOP UNITS	CODE BIT POSITIONS 76543210
000	1	8	1 or more	87654321
001	1	8	1 or more	87654321
010	1	7	1 or more	7654321x
011	1	6	1 or more	654321xx
100	1	5	1 or more	54321xxx
101	–	–	–	––
110	–	–	–	––
111	–	–	–	––

the High Level Keyer. A "zero" in this command bit allows serial data to be transmitted only to an internal data set (modem).

8.8 INTERFACE CONNECTOR

This interface is provided through an Amphenol 17-10500-1 connector. Pin assignements are as follows:

LEAD	FUNCTION	INPUT/OUTPUT
1	Protective Ground	––––
2	Protective Ground	––––
3	OH (Off Hook)	Output
4	+25v	––––
5	DA (Transmission Path Request)	Output
6	R (Ring Indicator)	Input
7	CCT (Data Coupler Ready)	Input
9	DT (4 wire)	Direct Private Line Connection
10	DT (2 wire)	(Direct Private Line Connection)
11	DR (2 wire)	(Direct Private Line Connection)
12	DR (4 wire)	(Direct Private Line Connection)
23	Clear to Send (RS-232)	Input
24	Transmitted Data (RS-232)	Output
28	Signal Ground	––––
29	Signal Ground	––––
32	+5v	––––
33	+5v	––––
40	Request to Send (RS-232)	Output
41	Received Data (RS-232)	Input
42	Data Terminal Ready (RS-232)	Output
44	Supervisory Transmitted Data (RS-232)	Output
45	Data Carrier Detector (RS-232)	Input
46	Supervisory Received Data (RS-232)	Input
49	Clock for 3300P	Output
50	Transmit Bit Rate Clock	Output

8.9 HIGH LEVEL OPTION

Interface with telegraph-type current loops is provided with the high level option. This option provides for completely isolated electronic neutral/polar output relay and a completely isolated neutral/polar input relay. Loop voltage may be as high as 400 volts across the relay and as high as 1000 with respect to ground.

Loop resistance and power is not included with the option.

For further information, refer to the Datapoint 2200 Installation Manual.

8.10 103-DATA SET OPTION CHARACTERISTICS

The 103-Data Set option provides for full duplex data transmission for rates up to 300 bits per second with a signalling system that is compatible with the Bell System 103 series Dataphones. Connection to the common carrier lines would normally be made through a Bell System Access Arrangement type F-58118, CBT, or 1001B. Other connections are also possible where automatic dialing or answering is not required.

The data set may be placed in either the answer mode or originator mode through the use of bit 6 of the communications adaptor command word (see paragraph 8.5). Bit 6 is set to 0 for answer mode and 1 for originator mode. The request to send command bit (bit 0) is normally set to 1 with the 103 option to maintain the transmit carrier on.

Operation of the automatic dialing and answering features discussed in paragraph 8.12 and 8.13.

Table 8-3 provides a summary of characteristics of the 103 Data Set option.

TABLE 8-3

103 DATA SET OPTION CHARACTERISTICS

Originate Mode Carrier Frequencies:	Transmit:	Mark: 1270 Hz Space: 1070 Hz
	Receive:	Mark: 2225 Hz Space: 2025 Hz
Answer Mode Carrier Frequencies:	Transmit:	Mark: 2225 Hz Space: 2025 Hz
	Receive:	Mark: 1270 Hz Space: 1070 Hz
Keying Rate:	Up to 300 bits per second	
Transmit Level:	0 to -10 dbm.	
Impedance:	600 ohms nominal	
Receive Sensitivity:	+5 to -30 dbm.	

8.11 202-DATA SET OPTION

The 202 Data Set option provides for either full or half duplex data transmission for rates up to 1200 bits per second (1800 bits per second on conditioned private lines). This option is compatible with Bell System 202 series Dataphones (including supervisory channel operation) and in addition provides a 150 bit per second supervisory channel when used with another Datapoint 2200 Data Set option of the same type. Connection may be directly to private lines or to common carrier lines through a Bell System Access Assignment type F-58118, CBT, or 1001B where access to the telephone switched network is desired.

Operation of the automatic dialing and answering are discussed in paragraphs 8.12 and 8.13 respectively.

Table 8-4 provides a summary of characteristics of the 202 Data Set option.

TABLE 8-4

202 DATA SET OPTION CHARACTERISTICS

Main Channel Frequencies:	Mark: 1200 Hz Space: 2200 Hz Soft Turn-Off: 880 Hz
Supervisory Channel Frequencies:	Mark: 387 Hz Space: 470 Hz Soft Turn-Off: 330 Hz
Special Command Frequency:	2025 Hz
Main Channel Keying Rate:	Up to 1200 baud (1800 baud on conditioned private lines.)
Supervisory Channel Keying Rate:	Up to 150 baud
Transmit Level:	0 to -10 dbm
Impedance:	600 ohms nominal
Receive Sensitivity:	+5 to -30 dbm

8.12 AUTOMATIC DIALING OPERATION

When using the Datapoint 103 or 202 data set options with the Bell System Access Arrangement type F-58118, CBT, or 1001B it is possible to automatically originate a call into the telephone switches network. The procedure for this function is as follows:

a. Set bits 4 and 6 of the communications adaptor command word to 1 to provide an off-hook signal to the telephone network and to prepare the modem for originate operation.

b. Test bit 7 of the communications adaptor status word for a 1 indicating dial tone present.

c. Set the transmitter time base to 100 bits per second (see paragraph 8.6).

d. Set the character length mask word to all zeros (ten bit length-see paragraph 8.7).

e. Set bits 3 and 7 of the command word to 1 thus inverting the serial transmitter output and transferring this output to the dial pulse keyer.

f. Sequentially transmit the octal byte 360 for each dial pulse required for each number (see paragraph 8.3-Data Output).

g. Program approximately 1 second delay between each number and at the end of the last number transmitted.

h. Re-establish the correct code length and bit rate for data transmission and set command word bits 3 and 7 to zero to restore the normal transmitter output.

8.13 AUTOMATIC ANSWERING OPERATION

When using the Datapoint 103 or 202 data set options with the Bell System Access Arrangement type F-58118, CBT, or 1001B it is possible to automatically answer a call from the telephone switched network.

Ringing is detected simply by testing bit 5 of the Communications Adaptor Status Word, Response to ringing would be to set bit 4 of the Communications Adaptor Command Word to 1 to provide an off-hook signal to the telephone network.

If the 103 Option is used Command Word bit 0 is set to 1 and bit 6 is set to 0 turning on the transmit carrier and selecting the answer-mode carrier frequencies.

If the 202 Option is used bit 5 of the command word is set to 1 for 1/2 to 3 seconds to transmit a 2025 Hz tone to disable echo suppressors and to inform the calling data set of out sequence in the telephone network, after which normal data transmissions occurs.

SECTION 2

DATAPOINT 2200

OPERATING SYSTEM

SECTION 2

THE OPERATING SYSTEM

The operating system is a conversational mode program for the Datapoint 2200 user to have a means to catalog, load, debug and run user programs and to provide other utilities important to the use of the 2200. All other programs discussed in the "Programmers Manual" such as the Program Editor and the Assembler are programs which the user may catalog onto an operating system tape and call into use as required.

The operating system itself is a relatively long program which is generally overlayed when user programs are called in from tape (unless they are less than 2K bytes in length and properly located). However, a family of resident utility routines is loaded with the operating system that may be used by user programs to simplify frequently used functions such as reading from the keyboard, writing to the CRT screen, reading and writing tape records, etc. The detailed use of the routines and the makeup of the operating system are described in Section 5—Advanced operating system command and subroutine usage.

Section 1 will describe the command language for the operating system and does not require any particular programming skills.

Start-Up Procedures

When power is first applied to the Datapoint 2200, it is incapable of performing any useful function except to load a block of data from the rear tape cassette deck into the processor's memory and transferring control to it. In the operating system, this first block of data is called a LOADER and when control is transferred to it, it proceeds to the first check itself to see if it was loaded properly and then to load the next file on the same tape which is the rest of the operating system program. This process can be executed at any time (assuming a proper program tape is in the rear deck) by pressing the RESTART key on the right hand side of the 2200.

This first block of data can be used to load programs other than the operating system and is generally useful for all applications of the 2200. In order to use the operating system, a full 8K bytes of memory must be provided in the 2200, but the loader alone can be used with any size memory.

When an operating system program tape is loaded, the first thing that appears on the screen is:

COMPUTER TERMINAL OPERATING SYSTEM

READY

At this point, any of the operating system commands discussed below may be typed into the 2200.

Each command has to be in the form of a word followed by the ENTER key, or a word followed by a space, or a dash and a name, or a modifier. Each of the following are valid operating system commands:

CATALOG
REPLACE RST4
REPLACE-RST4
RUN*

Only the first three letters of a command are actually decoded so that the following are valid commands:

CAT
REP RST4

If an invalid command is typed the system responds with:

WHAT?

Operating System Commands:

CATALOG

The CATALOG command will print out a list of programs currently available on that particular operating system tape. Up to 14 programs may be cataloged on a tape under this system and the order that they appear on the tape is the order their names appear on the screen when the catalog command is given. A typical catalog response might be:

CATALOG
RST4 CODER BANDIT ANNUIT
READY

Which would indicate that four programs are logged onto the tape.

NAME

The NAME command allows any program in the system to have its name changed. For example, the program named BANDIT in the above example can be changed to GAME with the command:

NAME-BANDIT, GAME

Names may have any combination of letters and digits up to six characters in length and beginning with a letter. All of the following are valid program names:

BOB
R12345
A
NAME

RUN

The RUN command causes the operating system to position the cassette tape to the program named in the run command, load the program into the 2200 memory and transfer control to it. The program being run may overlay part or all of the operating system. If it does, returning to the operating system can only be done by reloading it. This can be done by using the restart switch or by program control (See Section 5).

A run command would appear like this:

RUN-BANDIT

A program that has not been logged onto the system tape may be run by placing an assembled form of the program on the front deck and typing:

RUN*

IN

The IN command causes a program to be cataloged onto the operating system tape. The program must be assembled and the assembled program tape placed on the front deck. The IN command is typed giving the name to be assigned to the program as shown here:

IN RST4

The operating system tape (on the rear deck) will position itself to the end of its program library and will then copy the program from the front deck into place and add its name and position to the system catalog.

DELETE

The DELETE command causes the program named in the command to be removed from the system library. Unless the program being deleted happens to be the last program in the library, a SCRATCH tape will be required in the front deck to copy part of the library out and back to the system tape to CLOSE-UP the space. In this case, when the command is entered, the system will write a message back:

FRONT TAPE SCRATCH?

Then the processor will stop. The stop key on the right hand side of the keyboard will be lighted. If there is a tape in the front deck that can be recorded on, press the run key on the right hand side and the system will proceed to delete the named program for you.

This will generally take a little time. When the system is through it will write READY on the screen.

REPLACE

The REPLACE command allows a program already in the system catalog to be deleted and a new program to be put in its place in the same order on the tape. The new program does not have to be the same length as the old one. Again, this command takes time to execute due to the amount of "shuffling" of tape files to get everything in place.

AUTO

The AUTO command allows one of the programs in the system library to be marked for an automatic RUN whenever the operating system is restarted. Once a program has been named in an auto command, it may be cleared by typing a MANUAL command. The automatic feature may be overridden during a restart by holding down the KEYBOARD key on the right hand side of the keyboard.

The automatic program calling feature is particularly valuable when the program is to be run in an unattended situation. If the knock-out tab on the back of the operating system tape is removed then whenever power is reapplied or the processor is halted for any reason (including a programming halt), an automatic restart is executed and, of course, the program named in the auto command is reloaded and given control.

If the command AUTO is typed without a name then the system will respond with:

NAME REQUIRED

If a program is already named in an auto command then the system will respond with:

AUTO SET TO (PROGRAM NAME)

MANUAL

The MANUAL command will delete any program from the auto mode.

OUT

The OUT command causes any named program to be copied to the front tape causing any data already on the front tape to be lost. This copy may then be cataloged onto some other system tape or be saved for some future use. When using the OUT command the system will write FRONT TAPE SCRATCH? onto the screen and the processor will halt. If you have a usable tape in the front deck, then press the run key on the right hand side of the keyboard and the system will continue.

If the command OUT $ is typed, then the entire operating system including the library is copied to the front deck. If the command OUT * is typed, then the loader and the library is copied to the front deck but the operating system is deleted. This permits a program or family of programs to be used on a Datapoint 2200 with less than the full 8K of memory. (See Section 5 for details).

PREP

The PREP command causes the tape in the front deck to be rewound and a NULL file to be written at the beginning of the tape, effectively ERASING the tape and making it ready for use by operating system tape routines. The operating system commands that write on the front deck execute the PREP function automatically, however, USER PROGRAMS WRITING DATA TO THE TAPE MAY REQUIRE THE TAPE TO BE "PREPed" IN ADVANCE.

HEX

The HEX command allows programs generated on other machines that follow a specified hexadecimal format, to be loaded into the Datapoint 2200. Users will not normally be concerned with the HEX command. (See Section 5 for details).

DEBUG

The DEBUG command transfers program control to a small sub-program within the operating system that is used as a programming aid to debug and modify programs that are loaded into the 2200 memory. The debug sub-program allows you to write the contents of memory locations to the screen, modify memory locations, load programs into memory from the library or from the front deck, transfer control to parts of a program in memory and TRAP register values upon return to the debug program. Instructions on the use of the debug sub-program are given in Section 5.

SECTION 3

DATAPOINT 2200

SOURCE CODE EDITOR

SECTION 3

ASSEMBLER SOURCE CODE EDITOR

The assembler source code editor program provides for the preparation and editing of source data tapes in an assembler compatible format.

The editor program is called by the Computer Terminal Operating System (CTOS), if it is cataloged therein, by typing the following command:

STEP 1

RUN EDIT

STEP 2

a) When the editor has been loaded, the following message will appear on the screen:

Compressed Source Code Editor

Edit (E) or Convert (C)?

Type 'C' only if you have a tape generated by EDIT (1.1). This version of the Editor generated unblocked string records (using SSFW$). The present version generates "compressed source" records (using SNFW$).

If conversion is required, type 'C'. The following message will appear:

**PLACE SYMBOLIC TAPE IN FRONT DECK-
WHEN READY PRESS RUN**

Place the old tape in the front deck and push RUN. A converted file will be generated in the scratch area of the CTOS tape. When the 1st pass of the conversion is complete, the following message will appear:

**PLACE SCRATCH TAPE IN FRONT DECK-
WHEN READY PRESS RUN**

To protect your original tape, use a new tape to record the new compressed source code. Place the new tape in the front deck and press run.

The conversion process may be repeated for several tapes.

The Editor will now accept only compressed source tapes.

b) Type 'E' to EDIT, this message will appear on screen:

TYPE (:NEW,:OLD, OR:DUPLICATE)?

The meaning of the possible responses are as follows:

(NOTE:) All commands to the editor must begin with colons.

:NEW	Indicates that the tape on the front deck is to be treated as a new source data tape. Any old data on this tape will be written over by the editor program.
:OLD	Indicates that the front deck contains a tape with assembler source data on it. The operator will be allowed to edit this tape, changing only those lines which the operator specifies.
:DUPLICATE	Indicates to the editor that it should copy the contents of the scratch file to the source data tape. This provides copies of a single source data file. It also provides for recovery capability should a system failure occur during the editing process. See Step 4, Recovery Procedures.

STEP 3

a) If the response in Step 2 was :NEW the following question will appear on the screen:

NEW NAME?

The operator may now enter a character string of up to 40 characters. This text will become the first record of the source data tape and will appear as part of a comment line in the assembled text.

b) If the response in Step 2 was :OLD the editor will read the header record from the source data tape (front deck) and display that header on the screen in the following format:

OLD NAME IS XXXX XXXX...XXXX

This old name header is also written to the scratch file and is retained as the header record for the source tape.

c) If the response in Step 2b was :DUPLICATE the editor will return to Step 2b after the copying operation has been completed. See Step 4, Recovery Procedures.

After a) or b) above has been completed, a "READY" message will be written on the screen and the cursor will appear at the beginning of the bottom line. The editor is now ready to accept new text data or a command. In order to enter text, simply type the desired text. Upon pressing the enter key, the typed line will be rolled up on the screen one line and the cursor will reappear on the bottom line and accept another text line or a command. When a line of text rolls off the top of the screen, it is written to the scratch file. Lines will be written to the scratch file in the same order as they appear on the screen, the top line being first in the file.

Commands which can be entered, and their respective functions, are listed below. Command lines are distinguished from text lines by a leading colon; therefore, it is necessary to begin any command by typing a leading colon.

The "pointed line" referred to by some command descriptions below, is the line currently being pointed to by the visible pointer at the left side of the screen (col. 0). The "point's" vertical position is controlled by the keyboard/display keys. Specifically, pressing KEYBOARD causes the pointer to move up one line, pressing DISPLAY moves it down one line. Motion in either direction is circular around the screen. (It wraps around).

Commands may be entered from the bottom line only and must be preceded by a colon (:).

The functions available and their respective descriptions are listed below:

MANUAL SEARCH — (Not a typed command).
MANUAL SEARCH is like a continuing find of the very next line. That is, searching continues line at a time, with the next new line going to the eleventh line and the screen rolling up. It is unlike a find in that the screen isn't cleared for each new line acquired. It is useful for manually scanning through the data to bring to the screen and, therefore, into a position to edit the data of interest. MANUAL SEARCH is achieved by holding down the KEYBOARD and DISPLAY keys simultaneously. While held down, the search will proceed until end of file at which point the keys will become inoperable.

:FIND <TEXT> — Where <text> represents n characters of text data. The editor searches the source data tape for the first match with the desired n—<text> characters. When the desired line is found it is displayed on the bottom line of the screen. The search is circular through the data files. If no match is found the text which occupied the bottom line at the time the command was issued is restored. Leading blanks on data-lines are ignored during the search. A FIND or EOF in progress can be stopped by the manual search operation.

:COPY or :COP — The pointed line is copied to the bottom line and is simultaneously deleted from its previous location. The cursor will occupy the pointed line and will accept new text at this position. Striking the <enter> key returns the cursor to the bottom line and rolls the screen up one line.

:DELETE or :DEL — The pointed line is deleted from the screen and the scratch area. The cursor will occupy the pointed line and will accept new text at this position. Striking the <enter> key returns the cursor to the bottom line. The screen is not rolled up.

:INSERT or :INS — The pointed line and all lines above it are rolled up one line. The cursor will now occupy the blank line created and is ready to accept the new text. Striking the <enter> key returns the cursor to the bottom line.

:EOF — The editor will search the source data tape for an end-of-file. Upon finding it, the last 11 text lines are on the screen and the cursor occupies the bottom line ready to accept new text or commands. A FIND or EOF in progress can be stopped by the manual search operation.

:END — Causes the screen and the remaining source data to be copied to the scratch file. The scratch file is then copied to the source data tape.

:END/DEL — The same as :END except that all data on the source tape which follows the current screen data is deleted.

:SCRATCH or :SCR — Cause all lines between the top of the screen and the pointer, inclusive, to be deleted from the screen and the scratch area. The cursor will occupy the pointed line and will accept new text at this position. Striking the <enter> key returns the cursor to the bottom line. The screen is not rolled up.

STEP 4

Recovery procedures using the :DUPLICATE command can be implemented should the edit program be aborted during its execution without benefit of having completed all necessary copying and end of file writing.

Causes of difficulties which could require such action are:

1. Power failure during execution of the program
2. Turning off the power execution of the program
3. Striking the restart switch during execution
4. Encountering unrecoverable tape errors during execution
5. Removal of the tape cassette during execution

The EDITOR edits from one tape [the "Source Tape," containing old data] through the screen to the "scratch tape." The identity of the "source" and "scratch" tapes or files are reversed each time the current "source" tape reaches a file marker indicating end of file. Therefore, if an even number of passes have been completed, the updated file is on the front deck. If an odd number of passes have been made, the CTOS scratch file, file 40_8, is the updated file. New programs are written on the front deck.

Understanding the activity of the data with respect to the tapes will allow the operator to determine the course of action should difficulty occur.

Should the operator encounter difficulty and be rather vague as to which action to take, it is recommended that the operator choose the most valuable data tape by the following:

> When unrecoverable difficulty arises, remove the data tape and replace it with a new tape which will be written over by the scratch area of the operating system tape.
>
> Run the edit program and type :DUPLICATE in response to the original questions.
>
> The scratch area will be deposited on the new tape.
>
> Then the operator can manually search through the two tapes as separate data tapes and make a decision as to the most valuable to keep.

Special situations:

It is possible that the scratch area has the most valuable data on it but it is missing any file termination record. This can occur when the program was interrupted during data entry and the previous scratch-to-data tape copy (if any) was so far back as to render the scratch area the only desirable data. Should this occur, the copy will proceed from scratch to source until it runs out of data, in which case the operating system will encounter garbage on the tape. It will then write end-of-file marks on the data tape at that point. This will give a clean data tape suitable for beginning again. A note of caution: it is usually advisable, when attempting to recover valuable data, to use the two data tape approach and visually compare the two to make a value judgment as to which has the most desirable data.

SECTION 4

DATAPOINT 2200

ASSEMBLER

SECTION 4

THE ASSEMBLER

The 2200 assembly system consists of the ASSEMBLER, EDITOR and the OPERATING SYSTEM.

The ASSEMBLER generates a block of absolute object code which can be loaded by the operating system loader and cataloged by the operating system. It generates the object code from the symbolic source code which was generated by the editor.

The ASSEMBLER makes two passes over the source code.

The first pass generates a symbol table from the labels in the source code and checks for certain error conditions, primarily syntax and form. The symbol table is maintained in memory.

The second pass generates the program listing and the object code on the tape. It also produces further diagnostics of a more subtle nature.

Basically, the ASSEMBLER is a program that assigns numerical values to symbols and outputs these values upon input of the associated symbols. Symbols in certain fields have pre-assigned values such as the opcode mnemonics. The value assigned to an instruction mnemonic is the binary bit configuration recognized by the 2200 processor for that instruction.

For example, the following instruction mnemonics have the following octal values:

MNEMONIC	VALUE
ADB	201
RETURN	007
SUB	221

Symbols in fields other than the opcode field may be defined by the user. Pre-defined and user-defined symbols are kept separately by the ASSEMBLER so that the user may define symbols that are the same as the pre-defined symbols without encountering any difficulties.

Along with relating symbols with numbers, another major function of the ASSEMBLER is to enable one to reference a symbol that is defined later in the program. This is called FORWARD REFERENCING, and may be handled in a variety of ways. One of the simplest is to look at the source code twice. The first time determines the definitions of all the symbols and the second time uses the symbols to produce the object code. Each "look" at the source code is called a "PASS". Therefore, we end up with a two pass assembly process.

Statements

A 2200 assembly code statement consists of a label field, an instruction field, an expression field and a comment field. An example:

1	2	3	4
LABEL1	JTC	START	THIS IS THE COMMENT FIELD

Field 1 is the label field
Field 2 is the instruction field
Field 3 is the expression or operand field
Field 4 is the comment field

The 2200 editor provides automatic formatting so that the fields always are justified to begin in a certain column with tabbing to that field automatic. However, the ASSEMBLER only requires the following:

> A non-space in the first column means that the first field is a label, except for a leading period which designates the entire line as a comment line.
>
> A space in the first field means a null label and the first field is an instruction.
>
> Scanning proceeds from left to right with one or more spaces serving as field delimiters.
>
> Terminating fields by other than a space or a line termination will result in E-flags during the assembly.

THE LABEL FIELD may consist of up to 6 characters. An excess of 6 characters will be truncated. The first character may be any alphabetic character or a $ sign. The other characters may be any alphanumeric character or a $ sign. For example:

LEGAL	ILLEGAL
LABEL1	1LABEL (starts with numeric)
LABEL2	LABEL* (non-alphanumeric character)
LABEL$	LABEL. (non-alphanumeric character)
L1B2L3	

THE INSTRUCTION FIELD may be any of the instruction mnemonics listed in the Datapoint 2200 Reference Manual, compound instruction (described later) or assembler directives.

The Instruction Field may be from two to four characters. However, only the first three are scanned and consequently the user may abbreviate. For example:

LEGAL	ILLEGAL	
CALL	CALL2	(instructions have no more than four characters or numeric characters in the field)
JTZ		
SET		
TP		

THE EXPRESSION or OPERAND FIELD consists of any number of strings, numbers or symbols with operators between them. If a space or line end terminates a number or a symbol, the expression is assumed to be ended. Numbers are assumed to be decimal (base 10) unless they have one or more leading zeros, in which case they are taken to be octal. That is, 123 is 123 decimal, whereas 0123 or 00123 (the octal number 123) is really 83 decimal.

String quantities are delimited (preceded and followed) by apostrophies. In expressions, only the last character of a string is used if more than one appears. If a string were to be added to a number, only the last character of the string would be added. The character value is the ASCII binary number with the parity bit always a zero. A null string is legal (' ') and results in a zero value. The forcing character, #, is used in strings to indicate that the next character should be taken as ASCII no matter what it is. This is useful for getting the characters (') and (#) themselves into the string. For example:

' #' ##' is the character string ' #

There are three operators allowed in the expressions:

1. + This means addition
2. – This means subtraction
3. >8 This means shift right by 8. Use this to get the MSP of an expression.

Expressions are evaluated from left to right and all operations are assumed to have the same priority.

The operand or expression is a symbolic expression which is evaluated at assembly time and the value is used in whatever manner is required by the opcode.

THE COMMENT FIELD begins immediately after the first delimiter space after the operand. The comment field may have any character including punctuation within it. It is terminated by the end of the line which was written by the editor. Comments may take over the entire line, in which case that line must begin with a period.

ASSEMBLER DIRECTIVES are available for setting label and location counter values to other than the normal sequential location assignments and for defining constants. There are seven:

1. EQU — EQUALS. Sets the value of the label on the statement to the value of the operand expression.

2. SET — SET. Changes the value of the location counter to the value of the operand expression.

3. SK — SKIP. Increments the value of the location counter by the value of the operand expression.

4. TP — TABULATE PAGE. Increments the value of the location counter until it is a multiple of 256. This is useful for minimizing execution time and for blocking out data areas addressable by single precision.

5. DC — DEFINE CONSTANT. Generates eight bit object words from one or more expressions or strings following the opcode. If the expression is terminated by a space, the DC directive returns control to the main assembly process loop which obtains another instruction. If it is terminated by a comma, another expression or string is looked for. Another special exception is made for string items found in the DC directive. All the characters of a string item are significant and as many words as necessary are generated to accomodate all the characters of the given string. Again, a comma is looked for after the closing apostrophe in a string item to see if more expressions follow. This special string item is in effect only if the expression opened with an apostrophe. String items in expressions still have only one character of significance. For example:

   DC 1,2+3,2+'A','ABC'

   generates the following octal values:

   1,5,103,101,102,103

6. DA — DEFINE ADDRESS. Generates a two byte constant which is the address, LSP first, of the expression.

7. RP — REPEAT. Will cause the following line to be processed, the number of times, indicated by the operand value. For example:

   RP 5
   LDA 0123

   would produce the same code as:

```
LDA 0123
LDA 0123
LDA 0123
LDA 0123
LDA 0123
```

NOTE

Repeated statements which have a label on them will result in multiple definition of that label and all that entails, including the "D" error flags.

FORWARD referencing in the expression field in assembler directives only is not permitted.

8. END — END. Indicates that there is no more input data to be processed and that the ASSEMBLER should complete generating the output. The operand field has special significance in the END statement. The value of the expression in the operand of the END statement is the starting value of the execution of the program. That is the starting address. This is, of course, optional. When no operand is specified, the results are indeterminate. It should only be left vacant when the program is to be loaded without direct transfer of execution to the program such as an overlay.

Compound instructions are instructions which directly result in the assembly producing a sequence of source code. In this case, the 2200 ASSEMBLER has two: The HL instruction and the DE instruction:

1. HL LABEL — The HL compound instruction generates the load H-REGISTER and load L-REGISTER instruction necessary to place the address of the label LABEL in the H-REGISTER and L-REGISTER properly so that the load to and from memory will operate to that address. In doing the HL, it loads the most significant byte of the value of LABEL into the H-REGISTER and the least into L.

2. DE LABEL — The same as with HL except loads into the D and E registers.

THE ERROR FLAGS produced by the 2200 ASSEMBLER are as follows:

The error flags can occur during either pass of the ASSEMBLER in response to bad statements.

They are:

1. D — The D flag means DIFFERENT DEFINITION. It is flagged if the label has been redefined to a different value during the assembly. In that case, it has the second value.

2. I — The I flag means INSTRUCTION MNEMONIC UNKNOWN. The instruction was not an accepted instruction in which case a zero is inserted for this instruction.

3. E — The E flag means that an error has occurred in an expression or some unrecognizable character appeared in the wrong place. In this case a zero is substituted for the expression or in whatever was unrecognizable.

4. U — The U flag means UNDEFINED LABEL. It is used whenever a label is referenced and is not defined. This can occur in pass 1 when an assembly directive is operating on an expression containing a forward reference.

EXTERNAL COMMANDS & REFERENCES can be taken care of in two ways:

1. Directly produce the numeric value in the expression field corresponding to the reference external address (such as an operating system subroutine resident in memory) or the external command operand such as EX 1 instead of writing EX ADR.

2. Equating labels to these referenced locations using the EQU assembler directive and then referencing the labels. This can be done for external references to operating system subroutines by duplicating the operating system subroutine entry point label in your program and equating it to that address. i.e. instead of:

```
CALL 017000
```

to get the operating system keyboard string input routine, a more meaningful listing can be obtained if, at the beginning of the program, this was entered:

```
KEYIN$ EQU 017000
```

and then all references to this routine can be this way:

```
CALL KEYIN$
```

The same is true of the external commands used in the 2200. Rather than say:

```
EX 1
```

it is more meaningful to say:

```
EX ADR
```

Since it is an external command address that is desired.

A list of the external commands and the operands which the ASSEMBLER incorporates into the proper EX coding are below.

The ASSEMBLER treats external command labels differently to produce the octal command byte. For the commands, the operands are as follows:

ADR	1
STATUS	2
DATA	3
WRITE	4
COM1	5
COM2	6
COM3	7
COM4	8
UNUSED	9
UNUSED	10
UNUSED	11
UNUSED	12
BEEP	13
CLICK	14
DECK 1	15
DECK 2	16
RBK	17
WBK	18
UNUSED	19
BSP	20
SF	21
SB	22
REWIND	23
TSTOP	24

It is recommended that for those external commands used, the EQU to the table number is done at the start of the source program and then the external command references are done to the label.

Operating The Assembler

The ASSEMBLER must have a symbolic source tape generated by the 2200 editor.

Place this tape in the front tape deck.

Run the ASSEMBLER.

It will ask for printer speed. For the Datapoint 3300P, state 300. For a model 33 or 35 Teletype, state 110. For a model 37 Teletype, state 150. For no printer or no listing desired, state 0.

The source deck will rewind and begin to read in.

At the end of the first pass the ASSEMBLER asks if the second pass should proceed. It only requires a YES or NO. This is a convenience, since many times many errors will be uncovered by the ASSEMBLER already after the first pass and the user will desire to correct those errors before proceeding to the second pass and the listing.

If the second pass begins, the tape will rewind and begin accepting data again from the source tape, printing the listing and writing the object file on the scratch area of the rear tape.

When the tape has reached the end of the source the second time, assembly is complete and it only needs to copy the object code block on the rear tape to the area on the front tape just after the source code. This results in the rear tape being backspaced to the beginning of the block of code and then copying proceeding forward reading a block from the rear deck and writing it on the front deck.

At the end of ASSEMBLY, the operating system will be reloaded and come up running.

The front tape can be loaded into the machine to test using the operating system command RUN*, inputted into operating system catalog or loaded using the Debug program by using the F command.

SECTION 5

DATAPOINT 2200

ADVANCED OPERATING SYSTEM COMMAND and SUBROUTINE USAGE

1. INTRODUCTION

The primary function of CTOS is to provide the user with an easily accessible data environment which will greatly facilitate program generation. This function is fulfilled through the use of a file handling system which is available both directly from the keyboard in the form of system commands and through program calls to file handling input/output subroutines. Note that the keyboard facility deals mainly with the system (rear) tape (using the data (front) tape mainly for input/output and scratch space) but that the program routines are generalized to allow use of either tape.

1.1 KEYBOARD FACILITIES

The keyboard accessible facility allows the user to fetch and execute object files, which may be either system packages, such as the editor and assembler, or files the user has generated with either the assembler or other code generating programs. This facility also allows the user to create new files, alter or delete old ones, or perform certain utility functions. The system tracks the files on the system tape in a symbolic catalog which may be manipulated by the operator at the keyboard or used in program linking.

1.2 PROGRAM FACILITIES

The program routines perform basic operations such as reading and writing records with all parity checking and generation handled for the user. Other operations such as positioning to the beginning or end of a file, backspacing over records, or rewinding the tape are also provided. Parameterization is handled in a generalized way to make subroutine usage easy and consistent.

1.3 PHYSICAL LAYOUT

The memory layout of the operating system is shown below. The OS FILE HANDLER is the program accessible facility mentioned above while the OS COMMAND HANDLER is the keyboard accessable facility. Note that only 017400 and up need be in memory if only the symbol linker (which calls in an overlay by name so that its physical file number may be changed without having to rewrite the program calling in the overlay) is to be used, only 016200 and up need be in memory if only the debugging tool is to be used, and only 014000 and up need to be in memory if the keyboard facilities are not to be used (of course, 0-0777 is always reserved by the system). Also note that the user may load a program designed to fit into a 2K machine without overlaying any part of the full operating system.

CTOS MEMORY USAGE MAP:

	017777
SYMBOLIC LINKER	017600
CATALOG	017400
KEYBOARD DISPLAY	017000
DEBUG	016200
	016200
OS FILE HANDLER	014000
OS BOOTBLOCK COPY	013000
OS COMMAND HANDLER	05000
2K UNUSED	01000
LOADER	0

2. THE LOADER

The loader is the heart of CTOS. It enables other programs to load files from the tapes into memory without the tape having to be at the beginning of the desired file and provides extensive error protection. It is the routine used by the bootstrap mechanism (indeed, it is part of the bootblock) to load the initial program and is also the routine used in overlay and linkup operations both by CTOS and utility packages.

2.1 BOOTSTRAP ACTIONS

When a restart occurs, the rear deck is rewound and the first block on the tape (called the bootblock) is loaded into memory starting at location zero. The first 512 bytes of memory (0 to 0777 octal) have been reserved for a permanently resident program which is loaded from the bootblock. The first 40 bytes of this block constitute a program which runs a parity check on the rest of the block that should have been loaded. The processor is halted (note auto-restart implications if the auto-restart tab on the cassette is punched out) if this routine finds a fault in the check. Otherwise, zeros are stored in the memory locations used in the parity check routine. This will cause a halt if an early data drop-out from the tape machine occurs during the next bootstrap load (typically only one or two bytes get loaded in this failure mode). After the low memory has been cleared, a routine calls the loader, which has been loaded in the bootstrapping operation, asking for file zero to be loaded from the rear deck. If file zero cannot be loaded for some reason, the program halts the processor without a whimper (no bells or whistles in any of the bootstrap operation), otherwise, it jumps to the starting address supplied with file zero. Note that if the auto-restart tab is punched out of the rear cassette, any failure along the road of bootstrapping will cause the whole process to be tried again.

2.2 FILE ORGANIZATION

Once file zero has been loaded from the system tape, the bootstrap program (locations 0 through 074) is never used again until the next restart operation which will overlay it. The loader, however, will be used many times. The physical layout of information on the system tape is as follows:

BOOTBLOCK/FILE0/FILE1/. . ./FILE15/FILE32/FILE127

File 0 is the one executed by the bootstrap and is typically followed by a sequential (required to be sequential by the loader) set of minimally increasing (file numbers go up by only one at a time) files up to 15 (a CTOS catalog size limitation, although the loader will load a file with any positive number), followed by a file 32, which is a system scratch file, followed by a file 127 (largest positive eight bit number), which is a dummy to mark the logical end of the tape.

2.3 FILE LAYOUT AND RECORD FORMAT

Each file is a group of records starting with a very special four byte record. Every record used by CTOS starts with two special bytes to indicate that it is one of three types: file marker, numeric data, or symbolic data. The file marker, which is the special four-byte record at the beginning of a file, contains two additional bytes that denote the file number. The use of two bytes for both the record type and file number provides redundancy for error control, since the second byte is simply the one's complement of the first. The record types are denoted by 0201 for file marker, 0303 for numeric data, and 0347 for symbolic data. The following table summarizes all of the various data formats used by the system. XP and CP denote the two longitudinal parity checks and will be described later. FN denotes the file number and −FN its one's complement.

FILE MARKER RECORD: 0201 / 0176 / FN / −FN
NUMERIC DATA RECORD: 0303 / 074 / XP / CP / DATA
SYMBOLIC DATA RECORD: 0347 / 030 / XP / CP / DATA (with VRC)
FILE: FILE MARKER / DATA RECORD/ DATA RECORD / . . .
SYSTEM TAPE: BOOTBLOCK / FILE 0 / FILE 1 / . . . / FILE 15 / FILE 32 / FILE 127
DATA TAPE: FILE 0 / FILE 1 / . . . / FILE 127

2.4 LOADER ACTION

When the loader is told to load a given file, it begins searching the tape (the loader can load files from either deck, depending upon which entry point is used) forward until it finds a file marker record. Note that all records passed over must have a valid type number pair or an error recovery procedure will be initiated which will try up to three times to read the record correctly and then make an error exit if failure occurred all three times. Upon finding a file marker, the loader determines, from the number in that marker, whether the tape is positioned to the correct place (the number is equal to that requested), is not positioned far enough forward (the number is greater than that requested). If the tape is positioned to the correct place, the loader proceeds to load all of the numeric records it finds, obtaining the memory address of where it is to put the data from the beginning of each record, (symbolic records are ignored) until it runs across another file marker. At this point it stops the tape (which was in slew forward mode) and backs up over the file marker so a succeeding call on the loader would cause a file marker to be found immediately. If there were no numeric records in the file, an error return is made. If the tape is not positioned far enough forward, the loader searches forward for the next file marker. If the tape is positioned too far forward, the loader enters a reverse search mode. If, in this mode, the loader finds a file marker that indicates that the tape is now positioned to the correct place, tape motion is reversed and the file is loaded as in the forward search case. If it finds a file marker which indicates that the tape is not positioned backward far enough, the loader continues searching in the reverse mode for the next file. If, however, a file marker is found that indicates that the tape has been positioned too far backward, the loader decides that the file is not on the tape and makes an error return. Error returns are also made if a record can not be read without a parity failure or type indicator discrepancy (the two characters are either not the one's complement of each other or are not one of the three special numbers) occurring in all three trials or if loading the record would overstore the loader. In all of these cases, the carry condition will be true (a satisfactory load always rendering the carry condition false) and the tape will be positioned after any offending record.

2.5 PARITY CHECKING

The third and fourth bytes of every data record contain longitudinal parity checks. These bytes are set up by the record generation program such that the following exclusive OR sums will yield zeros: the first byte with all the data characters (data characters start with the fifth byte of the record and proceed to the end) and the second byte with the same characters except the sum is shifted right circularly one place after each exclusive OR. In the case of symbolic records, the additional condition of the vertical parity of each character being odd must also be met. One thing not mentioned in the discussion of the loader was that the first four data characters (fifth through eighth bytes in the record) are not really data but are the MSP and LSP of the starting memory address followed by the one's complement of the MSP and LSP of the starting memory address of where the data is to be loaded.

3. THE CATALOG, SYMBOLIC LOADER, BASIC I/O, AND DEBUG

As mentioned above, the operating system maintains a catalog of names which correspond to the files on the system tape. This catalog may be used in manipulating the files from the keyboard or in symbolically calling in overlays using the symbolic loader from a user program.

3.1 CATALOG CHARACTERISTICS

Each name in the catalog must start with a letter and may additionally contain from one to five alpha-numeric characters. There is room in the catalog for up to fourteen names so there is a limit of fourteen cataloged files on one system tape. The symbolic loader contains routines which will look up a given name up in the catalog and load the corresponding file. This same lookup routine is used by the command handler and is labeled LOOKUP.

3.2 UTILITY ROUTINES IN THE SYMBOLIC LOADER

Other utility routines in the symbolic loader area are a block transfer, labeled BLKTFR, and a routine, labeled INCSWP, which increments the H and L register pair and then swaps it with the D and E register pair. The block transfer will move the number of characters specified by the entry value in the C register from a memory address starting with the entry values in the H and L registers to a memory address starting with the entry values in the D and E registers.

3.3 LOADING ROUTINES

To use the symbolic loading routine, one loads into the D and E registers the address of the six characters of the desired name (trailing blanks must be included) and calls MLOAD$. If the zero condition is false upon return, then the given name was not in the catalog. If the zero condition is true but the carry condition is false upon return the loader could not either find or correctly load the file requested. Note that one must be certain to place the call to MLOAD$ in a place that will not be overlayed since execution will resume following the CALL instruction after the file has been loaded.

3.4 OTHER SYMBOLIC LOADER FACILITIES

Another facility in the symbolic loader area will load and execute a file whose number (not name) is in the B register upon call. Calling MAUTO$ will load the file from the system tape and calling MAUT2$ will load the file from the data tape. If the loader could not either find or load the file, the operating system is automatically reloaded.

3.5 KEYBOARD AND DISPLAY ROUTINES

The operating system contains facilities to ease the burden of communicating with the operator. Two routines exist. The first accepts the characters from the keyboard, displays them on the screen, and stores them into a memory buffer. The second writes a string of characters from a memory buffer onto the screen.

3.5.1 KEYBOARD INPUT

The keyboard input routine, labeled KEYIN$, accepts a specified maximum number of characters, given by the entry value of the C register, from the keyboard and puts them into memory starting at the entry value of the H and L registers and onto the screen at a starting horizontal cursor position of the entry value of the D register and vertical cursor position (which cannot be changed during the course of one input) of the entry value of the E register. Note that if the cursor collides with the right edge of the screen during entry, characters other than backspace, cancel, and ENTER will not be accepted, although they will print over each other in the last display position. The ENTER character (015) terminates input and is stored in the memory buffer to specify the end of data but is not written to the screen. Hitting the backspace key will delete the last character entered and move the cursor appropriately while hitting the delete key will delete all characters entered and also move the cursor appropriately. These two keys also back up the buffer memory pointer appropriately. Note that if one has typed a character at either the screen limit or at the maximum character count limit, hitting a backspace will cause the previous character to be erased and leave the last character still on the screen, although it will either not appear on the memory buffer or be after the 015.

3.5.2 DISPLAY OUTPUT

The display routine, labeled DSPLY$, will display the string of characters stored in memory starting at the address which is the entry value of the H and L registers and terminating with a character whose numerical value is either a 3 (ETX) or 015 (ENTER). The cursor starts at the entry values in the D (horizontal) and E (vertical) registers (a cursor position that is off the screen will not be sent to the CRT) and stops after the last character printed if the terminating character was a 3 or at the beginning of the following line if the terminating character was an 015. Note that, as in KEYIN$, the cursor stops at the right edge of the screen and the characters overwrite each other if more are available after collision. Also note that if display was occurring on the bottom line and the terminating character is an 015, then the whole screen is rolled up to force the existence of a following line and the information that was at the top of the screen is lost. After return from the display routine, the H and L registers will point to the location after the terminating character and the D and E registers will reflect the current cursor position. The cursor will be off while the display routine is writing, but it is turned back on upon exit even if it was off upon entry. Other special control characters can cause cursor positioning, line/frame erasure, and screen roll-up:

011 - a new horizontal position (0 to 79) follows
013 - a new vertical position (0 to 11) follows
021 - erase to the end of the frame
022 - erase to the end of the line
023 - roll the screen up one line

3.6 THE DEBUGGING TOOL

The debugging program allows the user to observe and modify any location in memory, to load files from either the system or data tapes, and to start execution at any place in memory. This allows him to load and debug programs with surprising ease. The major debugging technique is to insert RETURN instructions in critical places in memory so one routine at a time may be checked using the CALL command. All but two (user specifiable) of the registers may be saved upon return from the program being tested, allowing the user to determine if the proper actions are taking place by observing critical register and memory values. The registers A, B, C, D, E (subject to the H and L commands in Section 3.6.3) are stored in locations 16770, 16771, 16772, 16773 and 16774 respectively upon a return to Debug from a program which was called from Debug.

3.6.1 INPUT SYNTAX AND ERROR ACTION

The debugging program is entered from the command handler as explained in a later section or by processor execution control being passed to the location labeled DEBUG$. At this time the bottom line of the display will be erased and the current location and its contents will be displayed there. The program is now ready to accept input in the format <number><command>. The number is assumed to be octal and the absence of any digits between zero and seven implies a value of zero for the number. Only sixteen bits of significance are kept for the input value. If more are entered, the first digits entered are lost. Some commands use only the lower order eight bits. The number is terminated by the first character that is not between zero and seven and this character is taken to be the command. Note that leading spaces are not permitted. This line is read in using the KEYIN$ routine previously discussed, thus enabling the use of the backspace and cancel keys but requiring the ENTER key to be struck to obtain a response. In one case the ENTER character is the command and in some others the number is disregarded. If the command is not recognized, the program simply ignores it and the old current address and its contents are displayed again. After every command, control is returned to the entry point of the debugging program which will display the now current address and its contents.

3.6.2 THE CURRENT ADDRESS

Two memory locations in the debug contain an address (initialized to zero upon loading) which points to a memory location which is the current center of interest. Available commands allow one to change the contents of this memory location and move the pointer as well as perform other functions.

3.6.3 COMMAND MEANINGS

The following is a list of each command character with its effect and the number (in parenthesis) of bits of the given number used:

ENTER - set the current address to the number given (16)
I - increment the current address by one (0)
D - decrement the current address by one (0)
M - change the current address contents to the number given (8)
. - do the M followed by the I command (8)
L - upon return from a C command, cause the L register to be stored into the register whose number is given (3)
H - same as the L command but for the H register (3)
G - load from the system tape the file whose number is given (8)
F - load file one from the data tape (0)
0 - return to the operating system command handler (be sure it is there) (0)
C - execute a CALL instruction to the location whose number is given (16)

4. KEYBOARD FACILITIES (OS COMMAND HANDLER)

The operating system contains a program which will interpret user commands given at the keyboard and perform the tasks indicated. These tasks mainly involve copying new files from the data tape onto the system tape, copying files from the system tape onto the data tape, deleting and updating files on the system tape, and executing programs kept in these files, as well as several other functions.

4.1 SYNTAX RULES AND ENTRY ERROR ACTIONS

The command input format is purposely made quite strict to reduce the chance of causing unwanted action which could be catastrophic to the user's data. The command must start with the first character entered (leading spaces are illegal) and any alphabetic after the third character is ignored (thus DEBACLE will be interpreted as the DEB command just as well as DEBUG). The first non-alphabetic character must be either an ENTER, a space, or a dash (minus sign). Some commands will not allow the ENTER but typing a non-alphabetic other than these three will always net you an error message of WHAT?. This will also appear if a command that has legal syntax but is not one of those defined is entered. If the command is to be parameterized, the first name must follow the dash or space immediately and must be terminated with an ENTER if that is the only parameter. The name must start with an alphabetic but may contain any number of alpha-numeric characters even though all after the six will be ignored. If the command has two parameters, the first must be terminated by an ENTER. If a name is terminated by characters other than those specified, the error message BAD NAME will be displayed. If a name is not supplied but the command requires one, the error message NAME REQUIRED will be displayed. If the name given is required to be in the file catalog but is not, the error message NO SUCH NAME will be displayed. If the inverse is true, the error message NAME IN USE will be displayed.

4.2 OPERATING SYSTEM COMMAND INSTRUCTIONS

The following paragraphs describe the usage and effect of each command in the system. Each paragraph is titled by what must be entered to use the corresponding routine. Note that, for clarity, more than just the necessary three characters have been shown.

4.2.1 CATALOG

The CATALOG command lists the names of all files that are currently on the system tape. They are listed across the screen in the physical order in which the files appear on the tape. Any parameters supplied are ignored.

4.2.2 NAME (old), (new)

The NAME command will change the name of the file specified by the first name given to the second name given. This command requires that the first and not the second name be in the catalog. The catalog file on the system tape (a one record file (number one) that immediately follows the operating system file) will be overwritten with the new catalog. Note that this leaves the system tape positioned before the file marker of any existing first cataloged file. This operation is performed by all commands that change the catalog.

4.2.3 RUN (name)

The RUN command uses the loader to load the file specified by the name given and then transfers processor control to the starting address indicated to the loader by the file information. Note that it is the responsibility of the loaded program to return to or reload the operating system if this is desired. There is a special case to the RUN command that breaks the general syntax rules. If the name consists of exactly one asterisk terminated by an ENTER (RUN-*), the loader will be directed to load physical file 1 from the data tape. This provision is made to allow the user to run a program he has generated without having to load it onto the system tape. This, along with the F command in the debugging tool, eliminates a lot of tape movement when debugging programs.

4.2.4 IN (name)

Note that exactly the characters shown must be typed to execute this command since the space which must be the third command character will also terminate the command. This command will position the system tape after the last cataloged file and the data tape to the beginning of physical file 1. (The data tape convention is that physical file 0 will be the first piece of information on the tape, containing the users symbolic data for a given program, and that physical file 1 will be the second piece of information on the tape and will contain the users object data for a given program, and all tape after this is to be considered a scratch area which is properly terminated by physical file 127 to indicate the logical end of the tape.) The command then copies all records in the file from the data tape onto the system tape creating a file on the system tape (a file marker being written before the data was copied) which has the next available physical file number. Following this new file, file markers 32 and 127 are written on the system tape to indicate the new start of system scratch and logical end of tape. If the system tape contained no cataloged files before this command was issued, the file entered will be physical file 2 and immediately follow the catalog file. After the new file has been written, the new name is entered into the catalog and the catalog file is updated as in the NAME command. Note that if the catalog was full when the command was entered, the error message LIBRARY FULL will be displayed and no other action will occur. The name supplied must not already be in the catalog.

4.2.5 OUT (name)

The OUT command first executes the PREPARE command to provide itself with a null data tape which can be handled by the file handling routines. It then positions the system tape to the beginning of the given file (the name must have been in the catalog) and the data tape to the beginning of physical file one and copies all the records in the file on the system tape onto the data tape. It then places a file marker 127 on the data tape and quits. Note that the catalog file is not updated for this command. This command is provided to allow moving a file from one system tape to another through the associated use of the IN command.

There are two special cases to the OUT command that break the general syntax rules. If the name consists of exactly one dollar sign terminated by an ENTER (OUT-$) then an exact copy is made of the system tape up to file marker 32 at which time the copy is terminated by file markers 32 and 127 (which causes any scratch data on the old system tape to be removed). If the name is exactly one asterisk terminated by an ENTER (OUT-*), the action is similar to the previous case except physical files 0 and 1 (namely, the operating system) are deleted and the file numbers of all following data files (not file 32 or 127) are lowered by two. Note that if this tape is now bootstrap loaded, the first program loaded will be what was the first file cataloged in the operating system. This is most useful in preparing bootstrap tapes that will be used in machines with less than 8K of memory.

4.2.6 DELETE (name)

The DELETE command takes two different courses of action depending on whether or not the file deleted is the last one cataloged. If it is, the system tape is moved to the end of the next to the last cataloged file and file markers 32 and 127 are written, thus logically destroying the last file. The name is then deleted from the catalog and the catalog file is updated. If the file is not the last one cataloged, the PREPARE command is called to obtain a fresh data tape, as in the OUT command, and the system tape is positioned to the end of the named file. The rest of the system tape (up to the file 32 marker) is then copied onto the data tape and the data tape is terminated with a file marker 127. Note that the data tape file numbers start out at one and increase by one for each succeeding file copied onto the data tape. These numbers are not used since all the copy back part needs to know is file delimitation since it is getting its file number information from catalog positions. The copying onto the data tape is followed by the system tape being positioned to the end of the file before the one named and the data tape being positioned to the beginning of file one. A file marker having a value one greater than the previous marker is then written on the system tape and then the data tape is copied back onto the system tape with every file marker encountered on the data tape causing a file marker of value one greater than the previous marker to be written on the system tape. This process terminates when a file marker 127 is encountered on the data tape which causes file markers 32 and 127 to be written on the system tape. The given name is deleted from the catalog, all following entries are dropped down one place to correspond to the similar shift in file numbers that took place, and the catalog file is updated.

4.2.7 REPLACE (name)

The REPLACE command is quite similar to the DELETE command except that instead of preparing the data tape with the PREPARE command, it positions it to the end of file 1 and then writes a file marker 2. Now, copying all the files after the named one onto the data tape in a fashion similar to the DELETE command and copying the data tape back onto the system tape in exactly the same fashion as in the DELETE command will replace the named file by file 1 on the data tape, with any necessary physical expansion or contraction taking place. Even though the catalog is not changed in this operation, it is updated anyway since this is an easy way to position the system tape to a place before file marker 127. Without this, a succeeding call on the loader would run into trouble since the system tape would be left positioned after file marker 127 and the loader always starts by searching a tape forward which in this case would be off the logical end of the tape. The loader starts with a forward search because the very first time it is used, the tape is positioned just after the bootblock and a backward search for a file marker would cause trouble. The operating system routine which searches for files can start with a reverse search to avoid the problem since the tape will never be resting before file zero.

4.2.8 AUTO or AUTO (name)

There is a word in the catalog which contains the physical file number of a file which should be loaded and executed immediately upon loading and execution of the operating system. This enables a user program to be run after restart without interaction with the operation system being required. If this word is a zero or the keyboard switch is being depressed upon initial execution of the operating system, the normal entry is made into the operating system and the start up message and response request are displayed.

If the AUTO command is given with no name and the auto pointer is zero then the error message NAME REQUIRED will be displayed. Otherwise the name of the file being pointed to will be displayed in the message AUTO SET TO (name). If the auto command is given with a name (which must be in the catalog) and the auto pointer is a zero, the pointer will be changed to the corresponding file number and the catalog (which contains the pointer) will be updated. If the auto pointer is non-zero, the name is ignored and the AUTO SET TO (name) will be displayed as in the no-name case.

4.2.9 MANUAL

The MANUAL command will zero the auto pointer and update the catalog if the auto pointer was non-zero. Otherwise, the message AUTO NOT SET will be displayed.

4.2.10 PREPARE

The PREPARE command first asks the operator if the data tape contains anything of value and then halts. (Note that the auto-restart tab should not be broken out of the operating system tape because it will prevent use of the OUT, DELETE, or PREPARE commands since halting the processor will cause an auto-restart.) After the operator hits the RUN button as a response, it is assumed that the data tape is of no value as it is rewound and file markers 0, 1, and 127 are written on it. This is needed since the operating system routines require file markers for which they can search in using the data tape.

4.2.11 HEX (name)

The HEX command is similar to the IN command except that the data tape is formatted in symbolic records with no parity checking. This is useful in loading onto the system tape data produced by sources other than the 2200. There are four types of records accepted. The type is determined by the second character (the first must always be an 012 (LF)): asterisk means ignore the record; pound sign denotes the logical end of the tape; plus sign means the following four hexadecimal characters are a new starting address (these must be terminated by an 023 (XOFF)); and a hexadecimal character denotes a data record. All other cases are assumed to be data read errors. A data record must always contain an even number of only hexadecimal (0 through 9 and A through F) characters terminated by either an 023 or a plus sign. The characters are paired up to form successive bytes of eight bit data. If the terminating character is an 023 then the block of data bytes is written out in loader format and the starting address is incremented by the number of data bytes. If the terminating character is a plus sign then the data remains in the buffer and the following record will be appended to it. This allows blocks of larger than 36 bytes (128 is the upper limit) to be written when the device which writes the tape is limited to lines of 72 characters. Note that there is no buffer overflow protection and it is the responsibility of the program generating the symbolic data to keep the total number of continued bytes to 128 or less (128 hexadecimal character pairs). Also note that if a continuation line is followed by a new address line, the data will remain in the buffer but the starting address will change. This combination will cause incorrect results since even if the buffer did not overflow will also overwrite critical pointers which will cause the operating system to produce an error message (because it will be called with incorrect parameters when the critical pointers are overwritten) and be reloaded. If a read error is detected, the data tape is backspaced one record and read again. This will go on until the data appears correctly or the keyboard switch is depressed. Depression of the keyboard switch causes the same action as reading from the data tape a record starting with a pound sign.

4.2.12 DEBUG

The DEBUG command causes the debugging tool described earilier to be entered.

4.3 SYMBOLIC OPERATING SYSTEM AND EXTENDED COMMAND INSTRUCTIONS

The overlay program SOSX is available to extend the operating system command set. The following paragraphs describe the usage and effect of each new command. Each paragraph is titled by what must be entered to use the corresponding routine. Not that, for clarity, more than just the necessary three characters have been shown.

4.3.1 CHOP (name)

The CHOP command deletes the named file and all subsequent files.

4.3.2 INSERT (new, (old)

The INSERT command proceeds like a REPLACE command except it includes the old named file as one of the files written after file 1 on the front deck. When the front deck is copied back onto the CTOS tape a new object file has been inserted.

4.3.3 APPEND (name)

The APPEND command appends the object file from deck 2 onto the end of the named file on the CTOS tape. Like the DELETE command, it has two possible courses of action, depending on whether or not the file being appended is the last cataloged file. If it is, the tape is positioned to the end of the cataloged file and a new object file is copied from the front deck. New file 32 and 127 markers are written. If the named file is not the last cataloged file, the operation proceeds like REPLACE except that the CTOS tape is positioned to the end of the named file before the copy is performed.

4.3.4 LGO (name [, name, name . . .])

The LGO command makes a tape with a loader and the named file(s) in the sequence named in the command. The files will have sequential file markers starting with 0. There is a limitation of 23 characters on the command length, thus to name many files in the LGO command it may be necessary to temporarily rename the files with one character labels. LGO * is not permitted. OUT * has the desired effect of generating a load and go tape of all cataloged files.

4.3.5 SYMBOLIC (name)

The SYMBOLIC command adds a compressed source file (file #0) to the CTOS tape (in a fashion similar to IN). The name in the internal catalog will have an 'S' in the seventh (not displayed) position to identify the file as symbolic.

4.3.6 SREPLACE (name)

The SREPLACE, symbolic replace, command is performed exactly as the REPLACE except the compressed source file (file #0) is used instead of the object file. File 1 may be overwritten.

4.3.7 SINSERT (new), (old)

The SINSERT, symbolic insert, command is performed exactly as the INSERT except the compressed source file is inserted instead of the object file. File 1 will be overwritten.

4.3.8 ATTACH (name [, name, name . . .])

The ATTACH command positions the front deck to the end of file 0 and (without file markers) copies specified file(s) from the CTOS tape to the front deck. When all specified files are copied, the question 'END (LABEL OR :)?' will appear. A six character label may be entered. If ':' is typed no end statement will be added. The ATTACH * form of the command will attach, in cataloged sequence, all symbolic files to file 0 on the front deck.

5. PROGRAM FACILITIES (OS FILE HANDLER)

The operating system contains a set of routines which will perform all of the various input/output functions needed to maintain the files of data on the tapes. These routines are packed in the upper 2K of memory and are made available to the user if he wishes to handle his mass storage problems in conformance with the conventions of the operating system. All routines are uniformly parameterized and are accessed through an entry point table (a group of JUMP instructions to the actual routine locations) so any updates to the operating system will not have any effect upon the user's code.

5.1 ROUTINE PARAMETERIZATION

Routine parameterization consists of a memory location in the D (MSB) and E (LSB) registers of the first byte of a group of four bytes (called a packet) which parameterizes the call more explicitly. This method reduces the number of memory locations required to perform a routine call since, in a typical program, one needs only a few different packets but will have many different calls. The parameterization of some routines is not as extensive as that of others, but the same packet can generally be used for the different calls when they are affecting the same file.

5.1.1 LOGICAL FILE NUMBERS

The first byte in the packet is the logical file number and must be between zero and seven or an internal error H will occur upon calling any routine using this packet. This error condition usually occurs when the user has either failed to load the D and E registers at all or has loaded them with an erroneous value before calling the routine. The second and third bytes in the packet contain the LSB and MSB (respectively) of the first location in memory to be used as a data buffer. Actually, the two bytes previous to this location will be used by some of the routines as discussed later. This data buffer may be located anywhere in memory. The fourth byte in the packet specifies the length of the data buffer when numeric data is being handled. Note that using only one byte for the length implies that numeric records may not contain more than 256 data bytes. Actually, the maximum number of data bytes specified may not be greater than 254 for reasons that are made clear in the numeric routine instructions. The four bytes of the packet may be located anywhere in memory.

5.1.2 PHYSICAL DEVICE AND FILE NUMBERS

The logical file number specified in the packet is converted by each routine, via an internal transformation table, into physical file and device numbers. The physical device number specifies whether the operation is to be performed on deck on (rear) or deck two (front) and the physical file number specifies which file is to be treated on the given deck. Actually, not all routines use all of this information since, for instance, when one is reading records from a file he assumes that he is using the file to which the tape was last positioned. The internal transformation table is initialized at load time to the following values:

LOGICAL FILE	PHYSICAL FILE	PHYSICAL DEVICE	GENERAL USE
0	0	0	Unassigned
1	0	1	General deck one
2	0	2	General deck two
3	1	1	CTOS catalog
4	0	2	Symbolic data
5	1	2	Object data
6	0	0	Unassigned
7	32	1	System scratch

It is shown that logical files 1 and 2 are specified for use of any physical file, even though 0 is shown in the table. This can be done by use of a routine that will change the physical file number of a given logical file. A routine also exists to allow the physical device number to be changed, thus allowing the user to set up logical files in any physical configuration needed. Note, however, that one must have logical files 1 through 5 and 7 in the state shown (except for the physical device numbers of logical files 1 and 2) if one returns control to the operating system command handler, since the loaded values are assumed by this program. Logical files 0 and 6 may be used freely but must be set before the first call utilizing them. The following is an example of a packet usage as it would be expressed in the assembler: (Note all calls to CTOS tape routines must, as in the following example, be preceeded by a DE to the forst byte of the packet. Note also that the packet consists of 4 bytes: Logical file number, LSP of buffer, MSP of buffer and length of buffer)

taken in the case of fatal errors, for which it is decided that the only recourse is to reload the operating system. This is called an internal error and the message INTERNAL ERROR (letter) is written on the bottom line of the display before the system is reloaded. The various letters which may appear are the following:

- A - Illegal device specification
- B - Illegal record format
- D - Unrecoverable parity error
- G - Unfindable file
- H - Illegal logical file specification

The other path is non-fatal and simply returns with certain condition flags in states other than normal to indicate that something unusual happened. Since every routine uses a common subroutine (labeled GETPKT) to get the parameters from the specified packet, common internal errors can occur. If the logical file number is not between zero and seven, an

```
        LA      2         Set up logical file six to be used as physical file 3 on the
        DE      PACKET              front deck
        CALL    CPDN$
        LA      3
        DE      PACKET
        CALL    CPFN$
        DE      PACKET    Position to the beginning of the file
        CALL    PBOF$
LOOP    DE      PACKET    Read a record of symbolic from it into BUFFER
        CALL    SSFR$
        JTC     DONE      Quit if to the next file marker
        JFZ     TERR      Exit if type error
        .
        .                 Action taken for each record
        .
        JMP     LOOP
DONE    .
        .                 Action taken when file completely in
        .
TERR    .
        .                 Type error action
        .
PACKET  DC      6         Logical file 6
        DA      BUFFER    Buffer address
        DC      0         Length (not used)
        DC      0,0       Room for parity check generation
BUFFER  SKIP    128       Buffer area
```

5.2 ROUTINE USAGE INSTRUCTIONS

To use a routine, one sets up whatever is required for proper parameterization and then calls the desired location in the entry point table. The locations are labeled with the first word in the following paragraph titles followed by a $. For example, to call the serial numeric file read, one would say CALL SNFR$. The routine will either perform the requested task or take one of two error exit paths. The first path is

internal error H occurs. If the physical device number is not either a 1 or a 2, an internal error A occurs. Other than for these error actions, the following paragraphs described the effects of and the exact parameterization needed for each routine.

5.2.1 SNFR - SERIAL NUMERIC FILE READ

This routine reads the next record from the specified device. If the record is of type symbolic, the zero and carry conditions are set false and return occurs with no parity checking or data storage being performed. If the record is a file marker, the carry condition is set true and the tape is backed up to where it was before the routine was called. Again, return occurs with no data storage being performed. If the type is numeric, the two parity bytes followed by the data are read into the buffer. If the parity checking fails or the record type is bad, three efforts are made at reading the record by backing up to its beginning and starting over. If recovery is not made in one of these efforts, an internal error D occurs. If the record is read successfully, return occurs with the zero condition true, the carry condition false, and the H and L registers containing the memory location of the byte following the last one loaded from the tape. To calculate the length of the buffer area used, one must subtract the buffer starting address from returned values in the H and L registers. Remember that the first two characters in the buffer are not data characters but are the two longitudinal check sums. To obtain the number of data characters loaded, one must subtract the buffer starting address plus two from the returned values in the H and L registers. The parity checks are stored because the SBFW routine uses them instead of regenerating them from the data, thus shortening the time required to copy numeric records from one deck to the other.

5.2.2 SSFR - SERIAL SYMBOLIC FILE READ

This routine reads the next record from the specified device. If the record is of type numeric or file marker, the action taken will be the same as when SNFR reads a symbolic or file marker record. Action similar to that taken by SNFR is also taken if parity or type faults occur. If the record is read satisfactorily, only data characters will be in the buffer starting at the address specified. An 015 will mark the end of the data string and all vertical parity bits will be zero. The same normal exit conditions as in SNFR will occur.

5.2.3 SBFW - SERIAL BLOCK FILE WRITE

This routine writes a record of type numeric on the specified device. The total number of bytes, including the parity initialization sums as the first two, must be in the fourth byte of the packet. Note that inclusion of the parity initialization sums implies that the total number of actual data bytes cannot exceed 254. This routine assumes that the first two bytes in the buffer are the correct parity initialization sums since it does not generate them from the data. There are no error exits from this routine which implies that writing off the end of the tape will not be caught and that read-after-write checking is not performed.

5.2.4 SNFW - SERIAL NUMERIC FILE WRITE

This routine performs in a fashion similar to the SBFW routine except the two parity bytes are not included in the data buffer and the length specifies the number of actual data characters. The routine generates the two longitudinal parity sum initialization values and inserts them in the two locations preceeding the buffer. It then writes on the specified device a record of type numeric containing the two parity bytes generated, followed by the number of data bytes specified. Note that the length is adjusted to accomodate the two parity bytes so, as in the SBFW routine, only 254 actual data bytes may be written. If one specifies a length of 255 or 0, the only bytes (besides the record type) written on the tape will be respectively the first or both parity initialization sums. No error exits are made from this routine.

5.2.5 SSFW - SERIAL SYMBOLIC FILE WRITE

This routine performs in a fashion similar to the SNFW routine except that an 015 character in the data string rather than a specified value is used to determine the buffer length, vertical (in addition to longitudinal) parity generation is performed, and a record of type symbolic rather than numeric is written. The terminating 015 character is not included in the set of characters written to the tape, but remember that it will appear again if the SSFR routine is used to read the record.

5.2.6 PEOF - POSITION TO END OF FILE

This routine searches forward on the specified device until it finds a file marker. It then backspaces the tape until it is between the next to the last and the last record in the file. It then forward spaces the tape one record which puts it at the end of the file, having arrived there via forward tape motion. This forward arrival is important to observe when one plans to append one record after another and still maintain physical interrecord gap integrity. Note that every record passed over by the PEOF routine must have a valid record type or it will be read again in action similar to parity failure action in the SNFR routine.

5.2.7 PBOF - POSITION TO BEGINNING OF FILE

This routine searches for a file marker in a fashion similar to the loader except it starts by searching backwards. The file number searched for is specified by the physical file number supplied by the generalized parameterization. Note that since this routine starts by searching backwards, it will not decide that the requested file is not on the tape until it has found in the search forward mode a file marker that specifies a file number greater than the one desired, if indeed the file is not on the tape. Also note that if the leader is found in the search backward mode, the tape is positioned forward past the first record and the backward search is continued. If the first record is not a file marker (operating system convention requires it to be) or is a file marker whose value is greater than the one desired, the first record on the tape will be passed over back and forth until external intervention is imposed.

Otherwise, all search rules and error exit conditions of the loader routine apply here. If, upon return, the carry condition is true, then the file was not found. Otherwise, the tape will be positioned at the interrecord gap following the file marker, having approached that point with forward tape motion for the reasons expressed in the PEOF routine instructions.

5.2.8 BSP - BACKSPACE

This routine simply backspaces the tape one record using the hardware backspace function. No checking is made to see if the record was of proper type or if the tape ran onto the leader.

5.2.9 CPDN - CHANGE PHYSICAL DEVICE NUMBER

This routine stores the entry value of the A register (note the break from generalized parameterization) into the physical device number entry for the specified logical file in the internal transformation table.

5.2.10 CPFN - CHANGE PHYSICAL FILE NUMBER

This routine stores the entry value of the A register (note the break from generalized parameterization) into the physical file number entry for the specified logical file in the internal transformation table.

5.2.11 TRW - TAPE REWIND

This routine performs a hardware high speed rewind of only the front deck. If the rear deck (physical device 1) is specified, an internal error A will occur. Upon exit from this routine, the tape will be positioned to the clear leader.

5.2.12 TFNR - TAPE FILE NUMBER READ

This routine acts in a fashion similar to PEOF until it finds the file marker. At this point, it simply reads the value of that marker and leaves the tape positioned after the marker record. The value read is returned in the C register. Error exits similar to the PEOF routine can occur.

5.2.13 TFNW - TAPE FILE NUMBER WRITE

This routine will write on the specified deck the special four byte file marker record containing the physical file number specified. No error exits will occur.

SECTION 6

DATAPOINT 2200

TRACE

DATAPOINT 2200 TRACE PROGRAM

Introduction

TRACE is an interactive octal debugging aid for the Datapoint 2200. It operates under the Computer Terminal Operating System (CTOS) and occupies memory space between 5600_8 and 13777_8. The normally resident operating system subroutines are not overlayed and are callable by the program being traced.

TRACE accepts commands from the keyboard and displays its results in the rightmost eight columns of the CRT display. It allows a user to trace the execution of a program, to examine and change the contents of the registers and memory.

Entering Commands

TRACE commands consist of up to two octal operands followed by a single letter operation. If there are two operands, a comma shall separate the two. An operand may be a 13-bit address or an 8-bit byte value, either expressed in octal. If the operand is an address, it may be given in two parts, separated by a blank. The <u>first part</u> consisting of the five most significant bits, and the <u>second part</u> consisting of the remaining eight bits. An address may also be given as a single octal number, but it is displayed in two parts as described. Leading zeros need not to be entered.

Examples:

TYPED VALUE	DISPLAYED VALUE FIRST PART	SECOND PART
101A	–	101A
707J	01	307 J
0331, 1477W	–	331 ,
	03	077 W

The command being entered is displayed on lines 10 and 11 of the CRT display, as shown in Figure 1. Line 10 shows the first operand. Line 11 shows the second operand and operation. If there is only one operand, line 11 will be blank. If there are no operands, line 11 shows only the operation and line 10 will be blank. NOTE: In all ensuing examples, the display format is used to exemplify the referenced operation.

If an illegal character is typed, the beep signal will sound and the character will be ignore. The CANCEL key will cause the command being entered to be discarded and another command can be entered.

The ENTER key will cause the command just entered to be executed. The command may be CANCELed at any time before the ENTER key is depressed.

Command To Modify Registers Or Memory

A, B, C, D, E, H, L

The operations A, B, C, D, E, H or L take a single byte value operand. The register specified by the operation is set to the operand value.

Examples:

1 7 3 B	sets B to 173
0 0 1 A	sets A to 001
3 7 5 L	sets L to 375

Operation F

The operation F takes a single address operand. The Zero, Sign, and Parity flip-flops are set as if the lower 8 bits of the address were the result in A of some arithmetic instruction. The Carry flip-flop is set to the rightmost bit of the first part of the address.

Examples:

0 0 0 F	sets Zero resets Sign, Carry, and Parity
0 1 0 0 0 F	sets Zero and Carry resets Sign, and Parity
2 0 0 F	sets Parity and Sign resets Zero and Carry

Operation O

The operation O takes a single address operand. It opens the specified location for possible modification. The contents of the location are shown on line 12 of the CRT display. A byte value can be entered followed by ; , <,or> followed then by ENTER. The location is set to that value. If the terminating character is ; , TRACE will accept another command. If it is <, the previous location is now opened. If it is >, the next location is opened. If the CANCEL key is used, the currently open location remains open and any modification for it is discarded. The modifying byte value is shown as it is entered, following the contents of the open location on the CRT display.

Examples:

01 115 O	opens	01 115
<	now opens	01 114
>	reopens	01 115
57;	sets	01 115 to 057

00 017 O opens 00 017
5> sets 0 017 to 005, opens 0 020
20; sets 0 020 to 20

Command To Displayed Memory

Operation M

The operation M takes two operands, both addresses. They are the lower and upper bounds respectively of the region of memory to be displayed. Sixteen bytes are displayed across the entire width of the CRT display. The address is given on one line followed by the memory contents on a second line. The display continues being built and rolled up unless the DISPLAY key is depressed. The display then stops until the DISPLAY key is depressed again. The KEYBOARD key terminates the memory display.

Example:	000,	displays the first 256 bytes of
	00 377 M	memory

Transfer Of Control Commands

Operation K

The operation K takes one address operand. It causes a Call instruction to be performed to the address given as the operand. Return is to TRACE.

Example: 02 000 K calls routine at 2 000

Operation J

The operation J optionally takes one address operand. If the operand is absent, the content of the P register is used. It causes a Jump instruction to be performed to the address given as the operand or in P.

Examples:	03 101 J	jumps to 3 101 (octal 1501)
	J	jumps to address in P

Mode Setting Commands

Operation X

The operation X optionally takes two address operands. They are the lower and upper bounds respectively of a region in memory. Any Call instructions into this memory region are actually executed rather than being simulated. Since TRACE loses control at this point, it is imperative that routines called in this region return. All registers are properly set when the Call is performed. The contents of H and L are lost on the Return. If the operation X is given without operands, any region in effect is removed.

Example:	0 1 000 ,	sets the special CALL region to to 1 000 to 3 377
	03 377 X	(octal 400 to 1777)

Operation W

The operation W optionally takes two address operands. They are the lower and upper bounds respectively of a region in memory. At the completion of any instruction in this region, the registers contents are shown in the first nine (9) lines of the CRT display as shown in Figure 1. The contents of the Carry, Zero, Sign, and Parity flip-flops are shown as the letters C, Z, S, and P respectively on the right-hand side of the CRT display if set and blank is reset. If the operation W is given without operands, any region in effect is removed. If the regions for the X operation and W operation overlap, the X operation takes precedence.

Example:	12 000 ,	display the register contents after
	12 377 W	every instruction in 12 000 to 12 377 (octal 5000 to 5377)

Operation S

The operation S optionally takes one address operand. Before TRACE executes the instruction at the given address, the register contents are shown as in a W operation region and TRACE stops to accept commands. A J operation with no operands will restart the program. If the S command is given without an operand, any stop address in effect is removed. If the stop address falls within an X operation region, the X operation takes precedence.

Example: 7 011 S the program will stop before the instruction at 7 011 is executed

To stop a program when TRACE is in control, depress the KEYBOARD and DISPLAY Keys at the same time. This has the same effect as an S operation for the current program address.

If a Halt instruction is executed by TRACE, the result is the same, as if an S operation was set for the Halt otherwise ignored.

Starting Trace

TRACE is loaded like any other program from the operating system library (RUN TRACE). Once started, it will request the name of the program desired to TRACE. The name of the program must be typed, followed by ENTER. CANCEL will cause TRACE to ask again. The named program will be loaded using the symbolic linker and loader in the operating system, and TRACE will show the register contents as if the program had been stopped at its entry point. P and I will be the only registers with non-zero contents.

To TRACE a program already in memory, simply depress the ENTER key without entering a program name, then jump to the entry point of the traced program, using the J command.

To TRACE a program located on the front deck type * and depress the ENTER key. The object file (file #1) from the front tape will be loaded using the operating system loader, and TRACE will show the register contents as if the program had been stopped at its entry point. P and I will be the only registers with non-zero contents.

Operational Summary

A	Set A to operand
B	Set B to operand
C	Set C to operand
D	Set D to operand
E	Set E to operand
H	Set H to operand
L	Set L to operand
F	Set flip-flops from operands
O	Open location
J	Jump
K	Call
M	Display memory
X	Set special CALL region
W	Set register display region
S	Set stop address

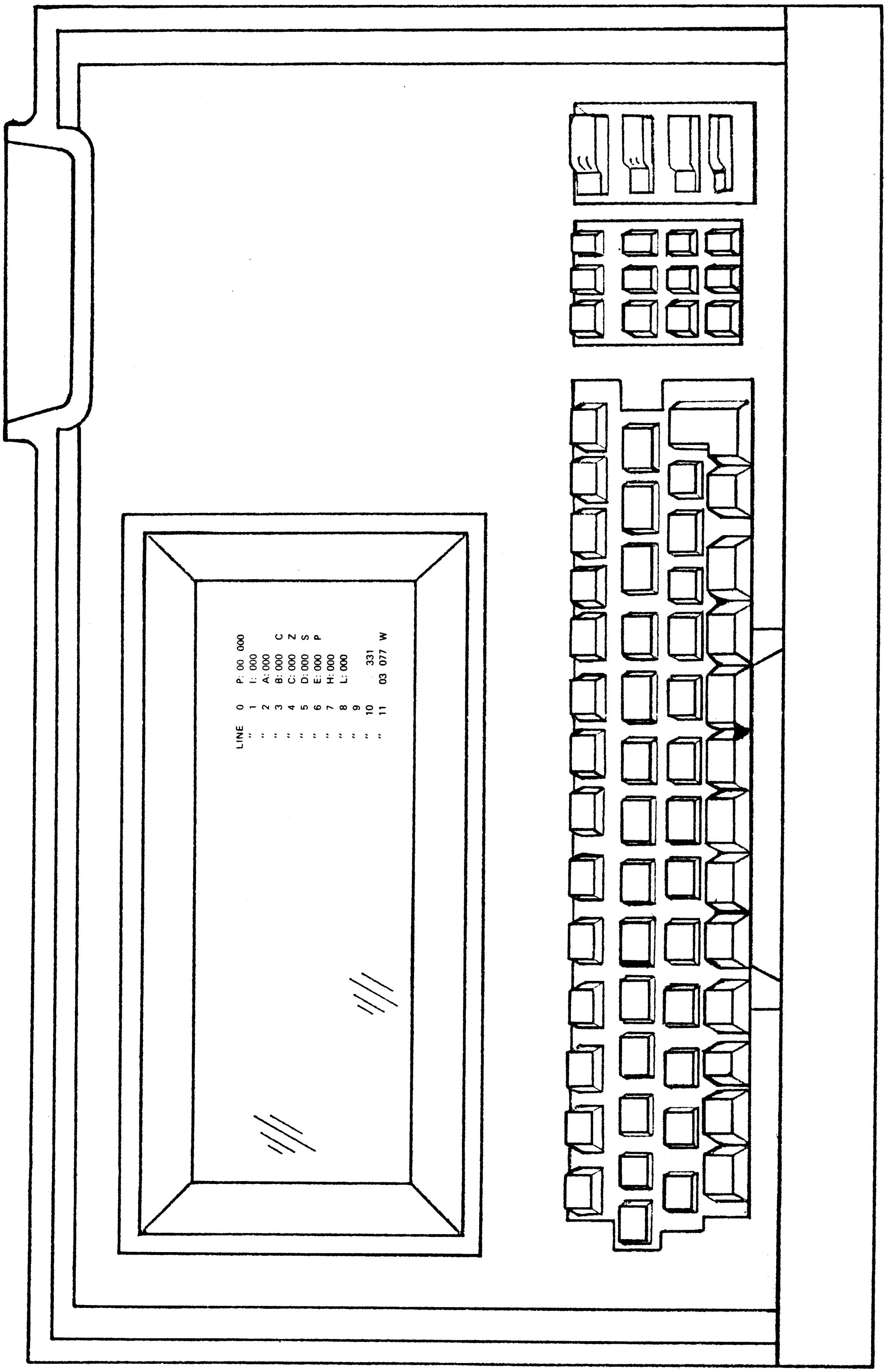


Figure 1. Operation W

SECTION 7

DATAPOINT 2200

ARITHMETIC SUBROUTINES

7.1.1 INTRODUCTION

STATH is a subroutine package specifically designed to provide formatted keyboard input, screen display, checksum and arithmetic operations on numeric strings. Each function of STATH is obtained by calling the entry point associated with that function.

Following is a list of the functions available through STATH. The labels given to their entry points and the sections incorporating their usage parameters:

Entry Point	Function
ADD$	Addition
COM$	Compare Magnitude
DIV$	Division
DSP$	Display on screen
KEY$	Keyboard formatted Input
MOD10$	MOD 10 checksum calculation
MOD11$	MOD 11 checksum calculation
MOV$	Move string
MUL$	Multiply
SUB$	Subtract

7.1.1.1 INTRODUCTION TO STRINGS - NUMERIC AND OTHERWISE

The purpose of a 'string' is to carry around a 'package' of text. A string is an individual block of text and just like a string it has a definite beginning and end. The composition of the string is an uninterrupted sequence of ASCII characters. That is, between the beginning and end of the string only ASCII characters are allowed. The ASCII character may be any of the 95 plus space (blank) characters listed in Section 1 of the Programmers' Manual (2200 Reference Manual).

The string is bounded at the beginning and end in different ways. The end is determined by the first occurrence of the ASCII 'ETX' whicn is equal to (003_8) in the sequence of characters called the string. The 003 tells STATH that the string is ended. The CTOS will also accept a carriage return character (015_8) in place of the 003 but STATH only accepts the 003.

The following are valid strings. The contents of the parentheses are intended to be the byte value of the ASCII character for single character values or the octal value of the octal triple such as 003.

(N) (O) (W) ( ) (I) (S) ( ) (T) (H) (E) ( ) (T) (I) (M) (E) (003)

(0) (1) (2) (3) (4) (5) (6) (7) (8) (9) (0) (003)

Which are in octal:

116,117,127,040,111,123,040,124,110,105,040,124,111, 115,105,003

and

060,061,062,063,064,065,066,067,070,071,003

Although a string has an inherent end built into itself, the 003, there is no beginning. At least no beginning which itself is part of the string of characters in memory. The beginning is combined with the pointer to the string itself. That is, a string is referred to by calling out a location in memory. That location is the first character of the string. In the above samples, for 'now is the time' to be referred to beginning with the word 'now' the location of the letter 'N' would be specified. It is clear that specifying only the 'N' yields a complete description which is:

'Begin with the character in the location specified and continue until a 003 is reached.'

Beginning with 'N' and continuing to the 003 gives: 'Now is the time'. If the location of the letter 'W' in now were specified, the string string resulting would be 'w is the time'.

Therefore, to specify a string to a routine (like STATH) which is going to use the string, the user must only transfer the address of the first character of the string or the character in the string the user wants to begin the string (it may not be the first) to the routine. Also, if the user created the string, he must be assured that there is a terminating 003 byte immediately following the last character of the string in memory.

STATH differentiates between two catagories of strings:

1) Numeric strings

and

2) Non-numeric strings

Where numeric strings are only regular strings with the character set restricted the characters 012345689 with an optional single period representing the decimal point and/or a single hypen leading the string representing a minus sign.

A non-numeric string is any string which is not numeric by the above definition.

A numeric string (ommitting temporarily the 003) can look like:

0000003456778888777.9999999999991

or

−123.45

or

34.5000000000

There is a size limit as to the number of characters a string may have in STATH. This is not true of ordinary text strings in CTOS where a string may, for some strange purpose, have thousands of characters in it. STATH is a mathematic package and the numeric strings represent numbers. The largest number of digits, therefore, is limited in STATH and that limit is 126.

7.1.1.2 INTRODUCTION TO THE FUNCTIONS OF STATH

STATH functions fall in the following four catagories. The catagories are listed with their appropriate functions below.

Arithmetic	Analysis	Manipulative	Input/Output
Addition	Compare	Move	Keyboard formatted input
Subtraction	MOD10 Check		Display on screen
Division	MOD11 Check		
Multiplication			

The arithmetic functions are the normal functions with which everyone is familiar.

The analysis functions permit decisions to be made on the content of a number. MOD10 and MOD11 verify the checksum Modulo 10 or 11 as is used in many business applications. Compare will permit comparing two numbers to determine equality or relative magnitude.

The move function is necessary as a preparation for using the multiplication and division functions in STATH, applicable for general use in the user's program to move numeric strings from one location to another and to format and round them in the process.

The input/output functions provide the user with simple techniques for bringing numbers into memory from the keyboard and displaying string numbers in memory onto the screen.

7.1.2 STATH FUNCTIONS AND ARGUMENTS

Each routine takes one or two arguments. An argument consists of a CTOS-compatable string. The argument strings are bounded at the end by an ASCII ETX (=003), and the beginning boundry is determined by the address contained in the register-pair associated with that argument. The maximum size for any STATH string is 126 characters. This means arguments and results are limited each to 126 digits.

Except for the routine DSP$, all strings must be 'numbers' which means a sequence only of ASCII numeric digits (0123456789) with an optional decimal point. Optional leading minus sign and optional leading blanks (an octal 040). The number must be right justified in the argument string. All strings except for DSP$ set the condition flags as follows:

Flag	Indication
Zero	The result was zero
Sign	The result was negative
Carry	An overflow occured
Parity	One or both arguments were improperly formatted

If parity is not set at the end of an operation, HL and DE contain the addresses of the location in memory past their respective ETX's. In the case of KEY$ and DSP$, D contains the column and E contains the row of the position immediately beyond the display area used. MUL$ and DIV$ leave D and E with junk in them. MOD10 and MOD11 leave H and L containing the address of the check digit position.

7.1.2.1 EXAMPLES ON THE USE OF STATH

Following is a 488-byte program which is a useful desk calculator using STATH. It is included as an example of a program calling STATH functions.

'DCLAC', the desk calculator, inputs a numeric string and provides addition, subtraction, multiplication or division of that inputted string against an accumulator. 'DCALC' always inputs the string from the keyboard into a string labeled 'input'. The accumulator is in a string labeled 'accum'.

The four arithmetic operations performed in the program are routines labeled as 'ADDOP', 'SUBOP', 'MULOP', and 'DIVOP'. The routines are very short but demonstrate the use of STATH.

```
         SET     01000
BOOT$    EQU     064
MOV$     EQU     010000
ADD$     EQU     010003
SUB$     EQU     010006
MUL$     EQU     06000
DIV$     EQU     06003
KEY$     EQU     010014
DSP$     EQU     010017
KEYIN$   EQU     017000
DSPLY$   EQU     017151
MLOAD$   EQU     017620
BEEP     EQU     13
HEADING  DC      021,811,20,2 2 0 0
         DC      013,5,011,31,'Total'
         DC      013,7,011,28,'Keyboard'
         DC      013,2,011,28,'0 To 9 '
         DC      'Decimal Places?'
DECPL    DC      '0',3
OVFMSG   DC      'Overflow',3
BLANK    DC      '          ',3
CLEAR    DC      022,3
INPUT    DC      '0000000000',3
ACCUM    DC      '00000000000',3
DIVID    DC      '0000000000000000000000',3
NAME1    DC      'Stath'
OPCODE   DC      '  ',815
DCALC    DE      NAME1
         CALL    MLOAD$
         JFZ     BOOT$
DCALCH   DE      0
         HL      HEADING
         CALL    DSPLY$
         LD      51
         LE      2
         HL      DECPL
         CALL    KEY$
         LL      INPUT
         CALL    FILLIN
         LL      ACCUM
         CALL    FILLIN
         LL      DIVID
         CALL    FILLIN
         LL      DECPL
         LAM
         SU      '0'
         LBA
         LC      '.'
         LA      INPUT+10
         SUB
         LLA
         LMC
         LA      ACCUM+10
         SUB
         LLA
         LMC
         LAB
         SLC
         LBR
         LA      DIVID+20
         SUB
         LLA
         LMC
         LD      28
         LE      2
         HL      CLEAR
         CALL    DSPLY$
         DE      ACCUM
         HL      ACCUM
         CALL    SUB$
DCALCL   LD      38
         LE      5
         HL      ACCUM
         CALL    DSP$
         LD      50
         LE      7
         HL      BLANK+6
         CALL    DSP$
         LE      38
         LE      7
         HL      INPUT
         CALL    KEY$
         LC      1
         LE      50
         LE      7
         HL      OPCODE
         CALL    KEYIN$
         HL      OPCODE
         LAM
         CP      015
         JTZ     ADDOP
         CP      'A'
         JTZ     ADDOP
         CP      'S'
         JTZ     SUBOP
         CP      'M'
         JTZ     MULOP
         CP      'D'
         JTZ     DIVOP
         CP      'E'
         JTZ     MOVOP
         CP      'R'
         JTZ     DCALCH
         EX      BEEP
         JMP     DCALCL
ADDOP    DE      INPUT
         HL      ACCUM
         CALL    ADD$
OVFTST   JFC     NOOVF
         LD      36
         LE      3
         HL      OVFMSG
         CALL    DSP$
         EX      BEEP
         JMP     DCALCL
NOOVF    LE      36
```

```
        LE      3
        HL      BLANK
        CALL    DSP$
        JMP     DCALCL
SUBOP   DE      INPUT
        HL      ACCUM
        CALL    SUB$
        JMP     OVFTST
MULOP   DE      ACCUM
        HL      ACCUM
        CALL    MOV$
        DE      INPUT
        HL      ACCUM
        CALL    MUL$
        JMP     OVFTST
MOVOP   DE      INPUT
        HL      ACCUM
        CALL    MOV$
        JMP     OVFTST
DIVOP   DE      ACCUM
        HL      DIVID
        CALL    MOV$
        DE      INPUT
        HL      ACCUM
        CALL    DIV$
        JMP     OVFTST
FILLIN  LAM
        CP      3
        RTZ
        LA      '0'
        LMA
        LAL
        AD      1
        LLA
        JMP     FILLIN
        END     DCALC
```

Observe the addition routine, 'ADDOP'. To add together the inputted string 'input' to the accumulator 'accum' the user only writes the following code as found at 'ADDOP'.

```
ADDOP   DE      INPUT
        HL      ACCUM
        CALL    ADD$
```

Executing this code will cause string 'input' to be added to the string 'accum' with the result in the string 'accum'. The accumulator, it must be realized, is simply a string which the writer of 'DCALC' is using as his result string and he preferred to call it an accumulator.

Note that after each operation there is a jump to 'OVFTST' or as in 'ADDOP', the code is immediately after and executed right after 'ADDOP'. Observe that the first instruction

```
OVFTST JFC NOOVF
```

of the overflow test is the actual test: If the carry isn't set then there was no overflow resulting from the operation. If the carry was set, in 'DCALC' the message 'overflow' is printed on the screen as is seen from the code following the 'JFC NOOVF'.

Subtraction behaves the same as addition except for the CALL to SUB$.

Multiplication and division are slightly different from addition and subtraction but operate similar to each other. Observe the following code as taken from 'DCALC'.

```
MULOP   DE      ACCUM
        HL      ACCUM
        CALL    MOV$
        DE      INPUT
        HL      ACCUM
        CALL    MUL$
        JMP     OVFTST
```

This demonstrates the requirement, as stated in 7.1.5, that, in MUL$ and DIV$, the argument #2 must be the result of the previous move. The reason for this is that multiplication and division really require three 'registers' or strings: The two strings being multiplied and the result. The 'MOV$' move operation makes a copy of whatever is being moved, during the move, in an internal STATH 'register' string. Therefore, note that the first three instructions in 'MULOP' cause the accumulator to be 'MOV$' moved to itself. Frequently the user can save time by utilizing this fact in making the last move before calling 'MUL$' a move of a string involving argument #2. (Again, argument #2 is the argument associated with the H and L registers).

Also note that 'MULOP' tests overflow using the same routine that is used for the other three arithmetic routines 'OVFTST' as described above.

7.1.3 LOADING STATH

STATH may be loaded in memory in either of two ways:

1) Incorporating the source code of STATH into the problem source code.

2) Catalog STATH as an object file and call it in through the operating system.

The second is preferred and simpler, as is done in 'DCALC'. Once cataloged, the following calls STATH into memory:

```
NAME1   DC      'STATH'
        DE      NAME1
        CALL    BOOT$
```

7.1.3 ADDITION

Entry Point Name	ADD$
Entry Point Address	10003 Octal
Argument #1 Address	D-E Registers
Argument #2 Address	H-L Registers
Result Location	Argument #2
Arithmetic Function	(Argument #2) = (Argument #2) + (Argument #1)

Action:
Adds two numeric string numbers, rounds, and installs leading blanks and trailing zeros when needed in the result.

Typical calling sequence:

```
ADD$    EQU     010003
.
        DE      ARG1
        HL      ARG2
        CALL    ADD$
```

Arguments:
Arguments must be each numeric strings of less than 126 characters in length. Argument 1 is addressed by the D and E Registers. Argument 2 is addressed by the H and L Registers and will contain the result.

Result:
The contents of argument 1 (D and E) will remain unchanged.
The contents of argument 2 (H and L) will contain the sum of arguments 2 and 1 and will have leading blanks and trailing zeros when needed.

Changes:
The contents of argument 2 are changed to contain the result.

Errors Recognized:
Improper argument format (parity bit set)
Overflow occurrence (carry bit set)

Comparison Flags:
Result was zero (zero bit set)
Result was negative (sign bit set)

7.1.4 SUBTRACTION

Entry Point Name	SUB$
Entry Point Address	10006 Octal
Argument #1 Address	D-E Registers
Argument #2 Address	H-L Registers
Result Location	Argument #2
Arithmetic Function	(Argument #2) = (Argument #2) - (Argument #1)

Action:
Subtracts one numeric string number from another, rounds and installs leading blanks and trailing zeros when needed in the result.

Typical calling sequence:

```
SUB$    EQU     010006
.
        DE      ARG1
        HL      ARG2
        CALL    SUB$
```

Arguments:
Arguments must be each numeric strings of less than 126 characters in length. Argument 1 is addressed by the D and E Registers. Argument 2 is addressed by the H and L Registers and will contain the result.

Result:
The contents of argument 1 (D and E) will remain unchanged.
The contents of argument 2 (H and L) will contain the difference of arguments 2 and 1 and will have leading blanks and trailing zeros when needed.

Changes:
The contents of argument 2 are changed to contain the result.

Errors Recognized:
Improper argument format (parity bit set)
Overflow occurrence (carry bit is set)

Comparison Flags:
Result was zero (zero bit is set)
Result was negative (sign bit is set)

7.1.5 MULTIPLICATION

Entry Point Name	MUL$
Entry Point Address	6000 Octal
Argument #1 Address	D-E Registers
Argument #2 Address	H-L Registers
Result Location	Argument #2
Arithmetic Function	(Argument #2) = (Argument #2) X (Argument #1)
Argument Restrictions	Argument #2 must be result of last MOV$ call

Action:

Multiplies two numeric string numbers, rounds and installs leading blanks and trailing zeros when needed in the result.

Typlcal calling sequence:

```
MUL$    EQU     06000

        DE      ARG1
        HL      ARG2
        CALL    MOV$

        DE      ARG1,
        HL      ARG2
        CALL    MUL$
```

Arguments:

Arguments must be each numeric strings of less than 126 characters in length. Argument 1 is addressed by the D and E Registers. Argument 2 is addressed by the H and L Registers and will contain the result. Argument 2 must have been involved in the previous move operation.

Result:

The contents of argument 1 (D and E) will remain unchanged.

The contents of argument 2 (H and L) will contain the product of arguments 2 and 1 and will have leading blanks and trailing zeros when needed.

Changes:

The contents of argument 2 are changed to contain the result.

Errors Recognized:

Improper argument format (parity bit set)

Overflow occurrence (carry bit set)

Comparison Flags:

Result was zero (zero bit set)

Result was negative (sign bit set)

7.1.6 DIVISION

Entry Point Name	DIV$
Entry Point Address	6003 Octal
Argument #1 Address	D-E Registers
Argument #2 Address	H-L Registers
Result Location	Argument #2
Arithmetic Function	(Argument #2) = (Argument #2) / (Argument #1)
Argument Restrictions	Argument #2 must be result of last MOV$ call

Action:

Divides one numeric string number into another, rounds and installs leading blanks and trailing zeros when needed in the result.

Typical calling sequence:

```
MOV$    EQU     010000
.
        DE      ARG1
        HL      ARG2
        CALL    MOV$
.
DIV$    EQU     06003
.
        DE      ARG1
        HL      ARG2
        CALL    DIV$
```

Arguments:

Arguments must be each numeric strings of less than 126 characters in length. Argument 1 is addressed by the D and E Registers. Argument 2 is addressed by the H and L Registers and will contain the result. Argument 2 must have been involved in the previous move operation.

Result:

The contents of argument 1 (D and E) will remain unchanged.

The contents of argument 2 (H and L) will contain the result of the division of argument 1 into argument 2 and will have leading blanks and trailing zeros when needed.

The number of decimal places in the result is equal to the number of decimal places in the dividend less the number of decimal places in the divisor. This number may not be negative and if it is, the number of decimal places is extended to make the difference zero.

The size of the result equals the size of the extended dividend less the size of the divisor.

Note that the string '10.0' divided by the string '3.0' is the string '3'. It is rounded to ZERO decimal places.

Changes:

The contents of argument 2 are changed to contain the result.

Errors Recognized:

Improper argument format (parity bit set)
Overflow occurrence (carry bit set)

Comparison Flags:

Result was zero (zero bit set)
Result was negative (sign bit set)

7.1.7 COMPARE

Entry Point Name	COM$
Entry Point Address	10011 Octal
Argument #1 Address	D-E Registers
Argument #2 Address	H-L Registers
Result Location	Arguments unchanged. Only sets condition code
Arithmetic Function	(cond-code) = (cond [(Argument #2) - (Argument #1)]

Action:

Compares two numeric string numbers as to magnitude. No change to arguments results. Changes are only made to the condition flags.

Typical calling sequence:

```
COM$    EQU     010011
.
        DE      ARG1
        HL      ARG2
        CALL    COM$
```

Arguments:

Arguments must be each numeric strings of less than 126 characters in length. Argument 1 is addressed by the D and E Registers. Argument 2 is addressed by the H and L Registers and will contain the result.

Result:

The contents of both arguments will remain unchanged. Only the condition code will change and will obtain the exact same condition as if a call to SUB$ were done. Therefore, the resultant condition flags will behave as if the result were to be rounded.

Changes:

The contents of both arguments remain unchanged.
Only the condition flags are changed.

Errors Recognized:

Improper argument format (parity bit set)
Overflow occurrence (carry bit set)

Comparison Flags:

Result was zero (zero bit set)
Result was negative (sign bit set)

7.1.8 MOVE

Entry Point Name	MOV$
Entry Point Address	10000 Octal
Argument #1 Address	D-E Registers
Argument #2 Address	H-L Registers
Result Location	Argument #2
Arithmetic Function	(Argument #2) = (Argument #1)

Action:

Replaces the numeric string number in argument 2 with that of argument 1, rounds and installs leading blanks and trailing zeros when needed in the result.

Typical calling sequence:

```
MOV$    EQU     010000
.
        DE      ARG1
        HL      ARG2
        CALL    MOV$
```

Arguments:

Arguments must be each numeric strings of less than 126 characters in length. Argument 1 is addressed by the D and E Registers. Argument 2 is addressed by the H and L Registers and will contain the result.

Result:

The contents of argument 1 (D and E) will remain unchanged.
The contents of argument 2 (H and L) will contain the number of argument 1 rounded and reformatted if necessary.

Changes:

The contents of argument 2 are changed to contain the result.

Errors Recognized:

Improper argument format (parity bit set)
Overflow occurrence (carry bit set)
[Note that overflow can occur in a MOV$ if a move from a larger to smaller field is attempted]

Comparison Flags:

Result was zero (zero bit set)
Result was negative (sign bit set)

7.1.9 MOD10 CHECKSUM CALCULATION

Entry Point Name	MOD10$
Entry Point Address	6006 Octal
Argument #1 Address	H-L Registers
Result Location	A-Register (no reformatting of argument)
Arithmetic Function	(A Reg) = Check-MOD-10 (Argument #1)

Action:
Checks validity of Modulo 10 checksum of a numeric string number.

Typical calling sequence:

```
MOD10$  EQU     06006
.
        HL      ARG1
        CALL    MOD10$
```

Arguments:
The argument must be a numeric string of less than 126 characters in length. Argument 1 is addressed by the H and L Registers.

Result:
The contents of the argument remains unchanged.
The carry bit is set if the check digit is 10.
The zero bit is set if the check digit is not 10.
The check digit is in the A-Register upon return.

Changes:
The contents of the argument remain unchanged.

Errors Recognized:
Improper argument format (parity bit set)

Comparison Flags:
Check digit was 10 (carry bit set)
Check digit was not 10 (zero bit set)

7.1.10 MOD11 CHECKSUM CALCULATION

Entry Point Name	MOD11$
Entry Point Address	6011 Octal
Argument #1 Address	H-L Registers
Result Location	A-Register (no reformatting of argument)
Arithmetic Function	(A Reg) = Check-MOD-10 (Argument #1)

Action:
Verifies the Modulo 11 checksum of the numeric string number.

Typical calling sequence:

```
MOD11$  EQU     06011
.
        HL      ARG1
        CALL    MOD11$
```

Arguments:
The argument must be a numeric string of less than 126 characters in length. The argument is addressed by the H and L Registers.

Result:
The contents of the argument remains unchanged.
The carry bit is set if the check digit is 11.
The zero bit is set if the check digit is not 11.
The A-Register contains the check digit.

Changes:
The contents of the argument remain unchanged.

Errors Recognized:
Improper argument format (parity bit set)

Comparison Flags:
Check digit was 11 (carry bit set)
Check digit was not 11 (zero bit set)

7.1.11 KEYBOARD FORMATTED INPUT

Entry Point Name	KEY$
Entry Point Address	10014 Octal
Argument #1 Address	H-L Registers
Extra Parameters	(D Reg) = Column. (E Reg) = Row for cursor
Input Function	(Argument #1) = (Keyed in number)
Input Restrictions	Screen format and, therefore, keyed in number has same format as originally in Argument #1

Action:

Provides formatted input from the keyboard into a numeric string. The format is maintained on the screen and only a number fitting the format can be entered. The inputted numeric string is placed in argument 1.

Typical calling sequence:

```
KEY$    EQU     010014
.
        LD      COLUMN
        LE      ROW
        HL      ARG1
        CALL    KEY$
```

Arguments:

The argument must be a formatted numeric string.
The D and E Registers must contain the column and row of the cursor position of the first character to be typed in.

Result:

The contents of argument 1 are replaced by the inputted number.
Striking the enter key with no input will cause the argument to be replaced with a zero.
The H and L Registers are pointing immediately after the ETX.

Changes:

The contents of the argument are replaced with the inputted string

Errors Recognized:

Improper argument format (parity bit set)

Comparison Flags:

Result was zero (zero bit set)
Result was negative (sign bit set)

7.1.12 DISPLAY STRING

Entry Point Name	DSP$
Entry Point Address	10017 Octal
Argument #1 Address	H-L Registers
Extra Parameters	(D Reg) = Column. (E Reg) = Row for cursor
Input Functions	(Display starting at D,E) = (Argument #1)
Input Restrictions	None. May even be non-numeric string

Action:

Displays a string onto the screen. String may be non-numeric.

Typical calling sequence:

```
DSP$    EQU     010017
.
        LD      COLUMN
        LE      ROW
        HL      ARG1
        CALL    DSP$
```

Arguments:

The argument may be a numeric or non-numeric string as long as it terminates with an ETX. The D and E Registers contain the column and row of the location of the first character of the string.

Result:

The string in argument 1 is displayed on the screen starting at the cursor location beginning with the column and row specified by the D and E Registers.
The H and L Registers point the location immediately after the ETX in argument 1.

Changes:

The contents of the argument remain unchanged.

Errors Recognized:

None

Comparison Flags:

None

7.2.1 INTRODUCTION

FPAK is a subroutine package which gives the Datapoint 2200 the capability of performing numerical operations with numbers in the range of -10^{-38} to 10^{37}. This is accomplished by representing all numbers in a form called "floating point." Floating point notation is a shorthand method of number representation and is very similar to the familiar "scientific notation" used in technical work.

FPAK also provides conversion of floating point numbers to and from 16 bit binary integers, particularly attractive for analyzing binary data gathered by the 2200 from instrumentation systems.

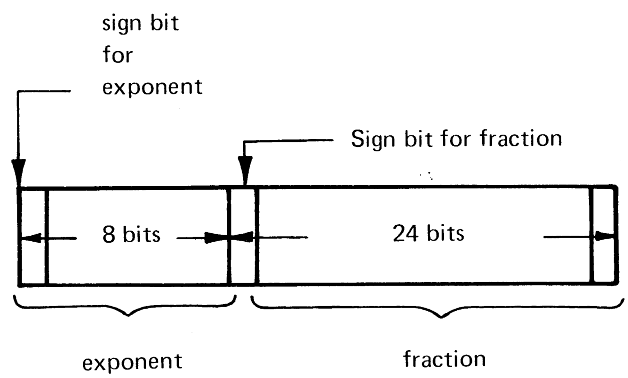
Supplied with FPAK is FCON which supplies the user with simple conversion of ASCII numeric string respresentation (suitable for displaying or printing) to and from the internal floating point representation.

The CTC 2200 floating point software consists of two main sections: FCON, the conversion section and FPAK, the arithmetic section. The conversion section converts a floating point number to an ASCII string, and visa versa. ASCII is the character code used by the 2200 keyboard and CRT display, and this section of the floating point software allows the user to interface with the computer. The user can enter numbers in a form familiar to him and read the results in a similar form. The ASCII string which the user enters through the keyboard (or from tape or some other means) is converted into the internal floating point form outlined in Section 7.2.1.1. When all arithmetic operations are completed, the user can request that the result be converted back into ASCII for display on the CRT (or for output to tape or for some other use).

7.2.1.1 INTRODUCTION TO FLOATING POINT REPRESENTATION

A number (N) in floating point form consists of two parts within the computer's memory--the "exponent" (e) and the "fraction" (f) --such that: $N = f*2^e$ (where the * denotes the multiplication operation). The exponent occupies one byte (word) in the 2200 and is an 8 bit signed integer. Thus, exponents on the 2200 can have a range of 127 to -128.

The fraction (sometimes called the "mantissa") on the 2200 occupies three bytes and is a 24 bit signed quantity. Like a decimal fraction (such as .5 or .0001), the fraction in a floating point number has a "decimal point," although "decimal point" is not what it is called. Its proper name, in a binary fraction, is a "binary point." In the notation used on the 2200, the binary point is located immediately to the right of the sign bit (high order bit) of the fraction. Thus, a picture of a 2200 floating point number would look like this:



The exponent and the fraction are separate parts of the number, and one can be positive while the other is negative. On the 2200, negative numbers are represented in their 2's complement form. Since the floating point representation requires more than one byte (word) on the 2200, a convention is used to address a floating point number in memory. The address of a floating point number is the byte (word) address of the exponent byte of the number. The software will use the addressed byte, and the three bytes immediately following, in whatever operation is being performed. Later in the documentation, reference is made to addressing floating point numbers. In such cases, this means that the MSP of the address of the exponent should be in the H or D register and the LSP of the address should be in the L or E register, depending upon whether the HL or DE pair is being used.

7.2.2 FCON - FLOATING POINT/STRING CONVERSION

7.2.2.1 INTRODUCTION TO FCON - FLOATING POINT/STRING CONVERSION

LOCATION	FUNCTION
FISC (0441)	Floating Internal to String Conversion entry point
FSIC (04444)	Floating String to Internal Conversion entry point
FSCE (04460)	Floating Set Conversion Error Branch entry point
OPER (013403)	Location of Floating point number to be converted to or from ASCII

7.2.2.2 FISC - FLOATING INTERNAL TO STRING CONVERSION

The Floating Internal to String Conversion routine has been designed so that the user need not specify the type of number he is going to supply. That is, as long as the ASCII characters being converted represent a valid, decimal number, the conversion routine can decide what type of number it is (i.e., integer, fraction) and perform the proper conversion without any further instructions. This type of input is referred to as "free form" input.

The result of all Floating Point Arithmetic routines end up at location 013403, labeled 'OPER'. The conversion routine, FISC, converts floating point numbers at OPER into a string beginning at the location specified by the H and L Registers upon execution of the CALL to FISC.

For example, should OPER (and the subsequent 3 bytes) contain the floating point number represented by 123,450,000,000,000,000,000 the string resulting from a call to FISC would look like this: 1.2345E20 where the ASCII number 1, an octal 61, would appear in the location specified by H and L and the period (an octal 056) in H and L plus 1 etc. A note of caution, FISC does not put a terminating 003 or 015 after the string. To be compatible with the CTOS string routines, the string must be terminated with either 003 or 015. However, FISC, upon return from being called, leaves the H and L registers pointing to the location immediately after the last character in the string. This enables the user to immediately store the terminating character of his choice (003 or 015) in that location upon a return. The following call to FISC will illustrate:

```
FISC    EQU     04441
.
        HL      String
        CALL    FISC
        LA      015
        LMA
```

Note the LA 015 and LMA will install a 015 as the terminating character to the resultant string which is the ASCII representation of the floating point number in OPER.

Name: Floating Internal to String Conversion (FISC)

Action:
Converts a floating point number to its ASCII character representation.

Calling Sequence:

```
FISC    EQU     04441
        HL      String
        CALL    FISC
        LA      015
        LMA
```

Arguments:
OPER contains the number to be converted to ASCII. The H and L registers contain the address of the location, in memory, where the first (leftmost) ASCII character should be placed.

Result:
The floating point number in OPER is converted to its ASCII representation, and the ASCII characters comprising this representation are placed in memory, beginning at the address specified by the contents of the H and L registers and continuing in sequential memory locations. H and L end up pointing to the next location after the last string character enabling the user to store the string termination character of his choice up on the return from FISC.

Changes:
The contents of OPER are destroyed; the previous contents of the output string are destroyed. At the end of the execution of this routine, H and L contain the address of the memory location immediately after the last ASCII character in the converted number.

Errors Recognized:
None.

Comments:
Numbers are represented to six significant (decimal) digits and are rounded where appropriate. The format of the output is "free," with small integer in FORTRAN I format, floating point numbers with decimal exponents between -6 and 6 in FORTRAN F format, and other numbers in FORTRAN E format.

7.2.2.3 FSCI - FLOATING STRING TO INTERNAL CONVERSION

The Floating String to Internal Conversion routine has been designed to convert floating point numbers into the proper ASCII representation. If the floating point numeric string is a small integer, it will be converted to an integer, with no decimal point in the representation. If the numeric string is a large integer, or a noninteger, it will be converted into scientific notation, or more precisely what is known as the FORTRAN E format, such as 1.3456E17.

FSIC converts to internal floating point representation an ASCII numeric string with optional leading minus sign, optional decimal point, and optional trailing FORTRAN E, type exponent, i.e. -1.2345E20. The H and L registers must point to the first character of the string. The result goes into the FPAK 'register' called OPER starting at 013403, ready to be used by FPAK. FSCE, Floating Set Conversion Error Branch, should be set first to cover format problems in the string being converted. A simple call to FSCE with the D and E registers specifying the location of your error recovery routine will set the error branch.

Name: Floating String to Internal Conversion (FSIC)

Action:
Converts an ASCII string, which represents a decimal number, into that number's floating point form.

Calling Sequence:

```
FSIC    EQU     04444
FSCE    EQU     04460
.
        DE      ERROU     Location of error
                          routine
        CALL    FSCE
        HL      String
        CALL    FSIC
```

Arguments:
The H and L registers contain the address of the first byte (character) of the ASCII string which represents the number to be converted.

Result:
The character string, if it represents a valid number, is converted to a floating point number, and that value is left in OPER. The result in OPER is normalized and rounded.

Changes:
The original contents of OPER are destroyed; the ASCII string is left unchanged, and upon successful conversion, the H and L registers contain an address of the character which caused termination of the number (i.e., was a character not allowed in the ASCII representation of a number).

Errors Recognized:
Invalid character found while converting from ASCII to floating point.

Comments:
The ASCII string may be in free form, that is, in FORTRAN I, F, or E format. All of those formats will be properly converted by this routine. Conversion stops when an invalid character (something other than a digit, "E", +, –, or .) is encountered <u>after</u> a valid number has been found. An invalid character encountered before a valid number has been found will generate an error. Some of the above characters can be considered invalid if used incorrectly (i.e., a "." in an exponent, such as 1.333E1.5, is an error) and will generate an error condition.

7.2.2.4 FSCE - FLOATING SET CONVERSION ERROR BRANCH

Name: Floating Set Conversion Error Branch (FSCE)

Action:
Specifies the location of the user's routine to be branched to in the event an invalid character is encountered while converting an ASCII representation of a number to the floating point representation of that number.

Calling Sequence:

```
        DE      ERROU     Location of error
                          routine
        CALL    FSCE
```

Arguments:
The D and E registers contain the address of the error routine.

Result:
The location in the floating point software which specifies the location of the error routine is set to the address provided by the user in the D and E registers.

Changes:
The previous error routine address is destroyed.

Errors Recognized:
None.

Comments:
In the event the user does not specify an error routine location, the floating point software will execute a return (RET instruction) if the error condition arises, and the arithmetic routine called by the user will continue to completion. At the conclusion of that routine, the contents of the A register will be non-zero, and the result in OPER will, in general, be erroneous. The user's error routine may end with a return if the user wishes to continue execution immediately after the call to the routine which generated the error.

7.2.3 FPAK - FLOATING POINT ARITHMETIC PACKAGE

7.2.3.1 INTRODUCTION TO FPAK - FLOATING POINT ARITHMETIC PACKAGE

LOCATION	FUNCTION
FCMP (04422)	Floating Point Compare
FADD (04400)	Floating Point Addition
FSUB (04403)	Floating Point Subtraction
FMUL (04406)	Floating Point Multiplication
FDIV (04411)	Floating Point Division
FLOD (04414)	Floating Point Load [memory to 'OPER']
FSTO (04417)	Floating Point Store ['OPER' to memory]
FNEG (04425)	Floating Point Negate [Two's complement]
FABS (04430)	Floating Point Absolute Value
FSTL (04463)	Floating Point Set Tolerance [For Equal Flag]
FFIX (04433)	Floating Point Fix [to 16 bit integer]
FFLT (04436)	Floating Point Float Conversion from 16 bit integer
FSOV (04447)	Floating Point Set Overflow Error Branch
FSUN (04452)	Floating Point Set Underflow Error Branch
FSDV (04455)	Floating Point Set Divide Check Error Branch

The second section of the floating point software is the arithmetic part. This section contains the routines for performing the common arithmetic operations of add, subtract, multiply, divide, compare, negate, and absolute value, and two routines for converting between integer and floating point formats (an integer, in the floating point software, is a 16 bit (2-byte) signed quantity which is addressed by specifying the address of the high order byte).

Within the floating point software package is a 4-byte area called OPER. OPER is to the floating point software what the A register is to the 2200 processor. Floating point operations are performed on numbers in OPER, or on pairs of numbers, one of which is in OPER and the other in memory. The software supplies two routines, FLOD and FSTO which provide the user with the capability of copying numbers from memory to OPER and from OPER to memory.

With two exceptions, all of the routines in the arithmetic part of the floating point software, which take floating point numbers as their arguments, expect their operands to be "normalized." Normalization is nothing more than an agreed upon standard for writing a floating point number. A number is considered normalized if the sign bit of the fraction and the bit immediately to the right of the sign bit (the high order bit of the fraction) are unequal. Thus, a positive fraction (sign bit 0) has a 1 as its high order bit, and a negative fraction (sign bit 1) has a 0 as its high order bit. This convention makes sure that the maximum precision possible is maintained in all floating point operations.

As a rule, all routines expect their floating point operands to be normalized. The significant exceptions to this rule are the add and subtract routines, FADD and FSUB. If the user is adding or subtracting two numbers, the numbers should be normalized for a result with the greatest accuracy possible. However, if the user has a floating point number which is not normalized, he can convert the number to its normalized form by adding or subtracting a "normal" 0 to or from the unnormalized number. A normal 0 has a fraction equal to 0 and an exponent of -128 (200 octal). Except in this case, it is not recommended that the user perform operations on unnormalized numbers.

7.2.3.2 ERROR CONDITIONS

There are several error conditions that can arise during the course of executing routines in the floating point software package. These errors are:

exponent overflow
exponent underflow
divisor of 0 (in FDIV)

For these errors, a flag (see below) is set to 1 when the error is detected. For all of these errors, an "error branch" is provided. When the error condition arises, the appropriate flag (or A register) is set, and a jump is made to a location in the floating point software package. This location contains a jump to the address of either a user-specified error routine or a return instruction (the default case if the user does not supply an error routine). There is a separate location for each error condition, and there are three routines -- FSOV, FSUN, and FSDV -- which are used to set or change the address of the error routines.

The error conditions and their respective flags are:

Exponent Underflow	UNFLO	Location 013400
Exponent Overflow	OVFLO	Location 013401
Divide by 0	DVDCK	Location 013402

If an error condition arises, the flag is set to 1 and a branch is made to the error routine address. If no error condition arises, the flag is set to 0, and a normal return from the routine occurs.

7.2.3.2.1 FSOV - FLOATING SET OVERFLOW ERROR BRANCH

Name: Floating Set Overflow Error Branch (FSOV)

Action:
Specifies the location of the user's routine to be branched to in the event an operation causes exponent overflow (the value of the binary exponent in the result is greater than 127).

Calling Sequence:
Execute CALL instruction location 04447.
See 7.2.3.2.

Arguments:
The D and E registers contain the address of the error routine.

Result:
The location in the floating point software which specifies the location of the error routine is set to the address provided by the user in the D and E registers.

Changes:
The previous error routine address is destroyed.

Errors Recognized:
None.

Comments:
In the event the user does not specify an error routine location, the floating point software will execute a return (RET instruction) if the error condition arises, and the arithmetic routine called by the user will continue to completion. At the conclusion of that routine, the appropriate error flag will be set to 1, and the result in OPER will, in general, be erroneous. The user's error routine should not end with a return since that would cause processing to continue in the floating point software with incorrect values in the machine registers.

7.2.3.2.2 FSUN - FLOATING SET UNDERFLOW ERROR BRANCH

Name: Floating Set Underflow Error Branch (FSUN)

Action:
Specifies the location of the user's routine to be branched to in the event an operation causes exponent underflow (the value of the binary exponent in the result is less than -128).

Calling Sequence:
Execute CALL instruction to location 04452.
See 7.2.3.4.

Arguments:
The D and E registers contain the address of the error routine.

Result:
The location in the floating point software which specifies the location of the error routine is set to the address provided by the user in the D and E registers.

Changes:
The previous error routine address is destroyed.

Comments:

In the event the user does not specify an error routine location, the floating point software will execute a return (RET instruction) if the error condition arises, and the arithmetic routine called by the user will continue to completion. At the conclusion of that routine, the appropriate error flag will be set to 1, and the result in OPER will, in general, be erroneous. The user's error routine should not end with a return since that would cause processing to continue in the floating point software with incorrect values in the machine registers.

7.2.3.2.3 FSDV - FLOATING SET DIVIDE CHECK ERROR BRANCH

Name: Floating Set Divide Check Error Branch (FSDV)

Action:
Specifies the location of the user's routine to be branched to in the event the divisor in a floating divide operation is 0.

Calling Sequence:
Execute CALL instruction to location 04455.
See 7.2.3.7.

Arguments:
The D and E registers contain the address of the error routine.

Result:
The location in the floating point software which specifies the location of the error routine is set to the address provided by the user in the D and E registers.

Changes:
The previous error routine address is destroyed.

Errors Recognized:
None.

Comments:
In the event the user does not specify an error routine location, the floating point software will execute a return (RET instruction) if the error condition arises, and the arithmetic routine called by the user will continue to completion. At the conclusion of that routine, the appropriate error flag will be set to 1, and the result in OPER will, in general, be erroneous. The user's error routine may end with a return if the user wishes to continue execution immediately after the call to the routine which generated the error.

7.2.3.3 FLOATING COMPARE

Name: Floating Compare (FCMP)

Action:
Compares, algebraically, two floating point numbers.

Calling Sequence:

```
FSTL    EQU     04463     Only necessary to EQU
FCMP    EQU     04422     once per program
.
        LA      TLRNC     Where TLRNC is the
        CALL    FSTL      comparison tolerance
.                         only necessary once per
                          program if tolerance
                          doesn't change
        HL      NUMBER    Number will be com-
        CALL    FCMP      pared with OPER
```

Arguments:
OPER contains one of the floating point numbers being compared, and the contents of the H and L registers address the other floating point number being compared.

Result:
Floating Compare sets the Sign and Zero flip-flops as if a subtraction of the floating point number addressed by the contents of the H and L registers from the floating point number in OPER had taken place. However, if the absolute value of the difference is less than or equal to the tolerance specified (see the description of the routine FSTL for an explanation of how the tolerance is specified), then the Sign and Zero flip-flops are set as if both floating point numbers were found to be equal.

Changes:
Neither operand is altered by the Floating Compare operation.

Errors Recognized:
None.

Comments:
Since representations of decimal fractions in a binary machine are approximate, the Floating Compare operation allows for an "approximate" compare by allowing the user to specify how close two numbers may be before they are considered equal.

7.2.3.4 FLOATING ADD

Name: Floating Add (FADD)

Action:
Adds two floating point numbers, rounds and normalizes the result.

Calling Sequence:

FSOV	EQU	04447	Only necessary to EQU
FSUN	EQU	04452	once per program
FADD	EQU	04400	
	DE	OVERR	Only necessary to set
	CALL	FSOV	these once per program or until it is desired to change.
	DE	UNERR	Where OVERR and
	CALL	FSUN	UNERR are addresses of user and recovery routines.
	HL	NUMBER	
	CALL	FADD	Number will be added to OPER

Arguments:
OPER contains one of the floating point numbers, and the contents of the H and L registers address the other floating point number.

Result:
The contents of OPER and the floating point number addressed by the contents of the H and L registers are added together with the result left in OPER.

Changes:
The contents of OPER are altered; the floating point number addressed by the contents of the H and L registers is unchanged.

Errors Recognized:
Exponent overflow, exponent underflow.

Comments:
Maximum precision is maintained by having both operands normalized; however, an unnormalized number may be converted to its normalized form by using this routine to add a "normal" 0 to the unnormalized number.

7.2.3.5 FLOATING SUBTRACT

Name: Floating Subtract (FSUB)

Action:
Subtacts two floating point numbers, rounds and normalizes the result.

Calling Sequence:
FSUB is identical to FADD except the program must now contain a FSUB EQU 04403 and the last statement in calling sequence is:

CALL	FSUB	Number will be subtracted from OPER

Arguments:
OPER contains the minuend and the contents of the H and L registers address the subtrahend.

Result:
The floating point number addressed by the contents of the H and L registers is subtracted from the floating point number in OPER, and the result is left in OPER.

Changes:
The contents of OPER are altered; the floating point number addressed by the contents of the H and L registers is unchanged.

Errors Recognized:
Exponent overflow, exponent underflow.

Comments:
Maximum precision is maintained by having both operands normalized; however, an unnormalized number may be converted to its normalized form by using this routine to subtract a "normal" 0 from the unnormalized number.

7.2.3.6 FLOATING MULTIPLY

Name: Floating Multiply (FMUL)

Action:
Multiplies two floating point numbers, rounds and normalizes the result.

Calling Sequence:
FMUL is identical to FADD except the program must now contain a FMUL EQU 04406 and the last statement in the calling sequence is:

CALL	FMUL	Number will multiply OPER

Arguments:
OPER contains the multiplicand, and the H and L registers contain the address of the multiplier.

Result:
The floating point of OPER and the floating point number addressed by the contents of the H and L registers are multiplied together with the result left in OPER.

Changes:
The contents of OPER are altered; the floating point number addressed by the contents of the H and L registers is unchanged.

Errors Recognized:
Exponent overflow, exponent underflow.

Comments:
This routine expects both operands to be normalized. If one or both of the operands is not normalized, erroneous results may occur.

7.2.3.7 FLOATING DIVIDE

Name: Floating Divide (FDIV)

Action:
Forms the quotient of two floating point numbers, rounds and normalizes the result.

Calling Sequence:

FSOV	EQU	04447	Only necessary to
FSUN	EQU	04452	EQU these
FSDV	EQU	04455	once per
FDIV	EQU	04411	program
	DE	OVERR	Only necessary to set
	CALL	FSOV	these once per program
	DE	UNERR	or when it is desired to change recover routine.
	CALL	FSUN	Where OVERR, UNERR,
	DE	CKERR	and CKERR are address-
	CALL	FSDV	es of user error recovery routines.
	HC	NUMBER	
	CALL	FDIV	Number divides OPER

Arguments:
OPER contains the dividend, and the H and L register contain the address of the divisor.

Result:
The floating point number in OPER is divided by the floating point number addressed by the contents of the H and L registers with the result left in OPER.

Changes:
The contents of OPER are altered; the floating point number addressed by the contents of the H and L registers is unchanged.

Errors Recognized:
Exponent overflow, exponent underflow, divisor equal to 0.

Comments:
This routine expects both operands to be normalized. If one or both of the operands is not normalized, erroneous results may occur.

7.2.3.8 FLOATING LOAD

Name: Floating Load (FLOD)

Action:
Copies a floating point number from its location in memory to OPER.

Calling Sequence:

```
FLOD    EQU     04414     Only necessary to EQU
                          this once per program.
.
        HC      NUMBER
        CALL    FLOD      Number is loaded into
                          OPER
```

Arguments:
The H and L registers contain the address of the floating point number that is to be copied into OPER.

Result:
The floating point number addressed by the H and L registers is copied into OPER.

Changes:
The original contents of OPER are destroyed. The floating point number addressed by the contents of the H and L registers is unchanged.

Errors Recognized:
None.

Comments:
None.

7.2.3.9 FLOATING STORE

Name: Floating Store (FSTO)

Action:
Copies a floating point number from OPER to memory.

Calling Sequence:

```
FSTO    EQU     04417     Only necessary to EQU
                          this once per program.
.
        HL      NUMBER
        CALL    FSTO      Number is loaded from
                          OPER
```

Arguments:
The H and L registers contain the address of the location, in memory, to which the floating point number is to be copied.

Result:
The floating point number is copied into the location addressed by the contents of the H and L registers.

Changes:
The original contents of memory (4 bytes) addressed by the H and L registers are destroyed. The contents of OPER are unchanged.

Errors Recognized:
None.

Comments:
None.

7.2.3.10 FLOATING NEGATE

Name: Floating Negate (FNEG)

Action:
Forms the two's complement of the floating point number in OPER.

Calling Sequence:

FNEG	EQU	04425	Only necessary to EQU this once per program.
.			
	CALL	FNEG	OPER is negated

Arguments:
OPER contains the floating point number to be negated.

Result:
The number in OPER is converted to two's complement form and then this result is normalized. The final result is left in OPER.

Changes:
The original contents of OPER are destroyed.

Errors Recognized:
Exponent overflow, exponent underflow.

Comments:
None.

7.2.3.11 FLOATING ABSOLUTE VALUE

Name: Floating Absolute Value (FABS)

Action:
Forms the absolute value of a floating point number.

Calling Sequence:

FABS	EQU	04430	Only necessary to EQU this once per program
.			
	CALL	FABS	OPER becomes the absolute value of OPER

Arguments:
OPER contains the floating point number whose absolute value is to be computed.

Result:
If the contents of OPER are greater than or equal to zero, then they are left unchanged. Otherwise, the contents of OPER are negated (see the description of FNEG). In the latter case, the original contents of OPER are destroyed.

Changes:
Contents of OPER are destroyed if they are less than zero; otherwise, the contents of OPER are unchanged.

Errors Recognized:
Exponent overflow, exponent underflow.

Comments:
None.

7.2.3.12 FLOATING FIX

Name: Floating Fix (FFIX)

Action:
Converts a floating point number into a 16 bit integer.

Calling Sequence:

FFIX	EQU	04433	Only necessary to EQU this once per program
.			
	HL	NUMBER	Number and number+1
	CALL	FFIX	will contain the 16 bit integer made from OPER

Arguments:
OPER contains the floating point number to be fixed (converted to an integer), and the H and L registers contain the address, in memory, of the high order byte (upper eight bits of the integer) where the integer is to be placed.

Result:
The floating point number is converted to a 16 bit integer. If the number has a fractional part, that part is lost. The 16 bit integer is stored in memory beginning at the byte addressed by the contents of the H and L registers.

Changes:
The original contents of the 16 bits addressed by the contents of the H and L registers are destroyed. The contents of OPER are unchanged.

Errors Recognized:
None.

Comments:
If the number in OPER is such that it cannot be represented in 16 bits, only the low order 16 bits are stored in memory. Any higher order bits are lost.

7.2.3.13 FLOAT

Name: Float (FFLT)

Action:
Converts a 16 bit integer into a normalized floating point number.

Calling Sequence:

FFLT	EQU	04436	Only necessary to EQU this once per program
.			
	HL	NUMBER	The 16 bit integer in
	CALL	FFIX	number and number+1 will be converted to floating point in OPER

Arguments:
The H and L registers contain the address, in memory, of the high order byte (high order eight bits) of the 16 bit integer that is to be converted.

Results:
The 16 bit integer is converted from its integer form to the floating point form, and the result is normalized and left in OPER.

Changes:
The original contents of OPER are destroyed. The 16 bit integer addressed by the H and L registers is unchanged.

Errors Recognized:
None.

Comments:
None.

7.2.3.14 FLOATING SET COMPARE TOLERANCE

Name: Floating Set Compare Tolerance (FSTL)

Action:
Specifies a range in which the difference of two floating point numbers must lie for the two numbers to be considered equal.

Calling Sequence:
See 7.2.3.3.

Arguments:
The A register contains the tolerance as a positive eight bit integer (the high order bit of the A register must be 0).

Result:
The location in the floating point software which specifies the floating point compare tolerance is set to reflect the value provided by the user in the A register.

Changes:
The previous value of the tolerance is destroyed.

Errors Recognized:
None.

Comments:
When the floating point package is initialized, the tolerance is set as if the user had called FSTL with a 2 in the A register. If the value in the A register is less than .0 when FSTL is called, erroneous results may occur when using the floating compare routine, FCMP.

SECTION 8

DATAPOINT 2200

COMMUNICATIONS SUBROUTINES

1. INTRODUCTION

Interfacing the Datapoint 2200 with a wide range of communication facilities is a simple task. All that is needed is the 2210 ACA Communications Adaptor with the required data set or keyer option and the necessary software subroutines to drive it. The software subroutines may or may not have been written for a particular application. However, it seems likely that most users will choose to develop their own to fit their particular needs. This chapter is devoted to aiding the user in fulfilling this goal.

Understanding communications subroutines is useful for many reasons. It enables use of communication disciplines not previously used to fill a specialized need. It enables a user to develop routines that are most efficient for his particular application, it permits a user to modify previously written routines for special purposes and provides greater insight into how the communications system functions.

There is nothing difficult about the communications routines. They are just another part of the user's applications program. The routines are given special treatment here because they are used so frequently and because the terminology and hardware used for communication is foreign to many users.

In addition to the material covered in this chapter the user should be familiar with material covered in other publications on the subject of data communications. Two references that are highly recommended before embarking on any communications oriented 2200 applications are:

Bell System Data Communications Technical Reference Manual*

Martin, James; Teleprocessing Network Organization; Prentice - Hall, 1970

2. TYPES OF SUBROUTINES

As in most modern computers, the input/output devices used with the Datapoint 2200 are much slower than the 2200 processor. In order for an input/output (I/O) routine to be efficient it must be possible for the processor to perform other tasks (including other I/O operations) while a given I/O routine is active. One approach is to use an interrupt system in the processor to stop one routine and give control to another when an I/O operation is needed. The Datapoint 2200 does not have an interrupt system but in its place it has a very powerful subroutine calling mechanism which permits many separate I/O routines to be "scanned" during normal execution of a program so that several I/O or other subroutines can be active at the same time.

This leads to the two possible types of communications subroutines: "in-line" and "interleaved". In-line subroutines are those routines which are written in such a way that whenever they are called they "capture" the processor until their function is complete and hence do not permit any other subroutine to be active at the same time. In many situations in-line subroutines are all that is required (such as during an automatic dialing operations when the 2200 has no other functions to perform). Interleaved subroutines are written in such a way that they return to the calling routine at regular intervals while they are active - to be called again to complete their work. Return points in communications subroutines frequently occur following status checks of external devices so that the communications subroutine does not sit in a "tight-loop" waiting for some external operation to be completed.

All of the I/O routines in the CTOS (Operating System) are in-line and would not be used during interleaved operations.

3. INPUT/OUTPUT OPERATIONS

In order to write any type of I/O subroutine for the Datapoint 2200 it is necessary to have a working knowledge of the input/output section of the processor. All 2200 I/O devices (including the CRT, keyboard and tape cassette decks as well as the Communications Adaptor) operate alike and have the same general I/O structure.

The basic physical details of the I/O structure are given in Part 4 of the Datapoint 2200 Reference Manual. We will deal with this system here from a programmer's point of view.

3.1 Data Buses

Data flow to and from the processor takes place over a set of I/O data lines connected to the A-register in the processor. Output data is transmitted from the A-register by eight wires which at all times reflect the contents of the A-register. Whenever the content of the A-register is to be transmitted to an I/O device, one of the external command instructions is executed, which causes one of the External Command Strobes to pulse a signal to the I/O device, informing it that the data on the output bus is for it, and should be read.

*Obtained through Engineering Director—Data Communications, American Telephone and Telegraph Co., 195 Broadway, N.Y., N.Y. 10007

Input data is transmitted to the A-register in the processor by eight wires which form a bus connected to all I/O devices. Each I/O device is so arranged that only the one currently addressed will have access to this bus. Normally, when an I/O device is first addressed, a status word is placed on this bus. The status word (or whatever is placed on the bus) is loaded into the A-register whenever an INPUT instruction is executed.

3.2 External Command Strobes

The Datapoint 2200 processor has 24 External Command Strobes in its I/O structure, only eight of which are brought to devices outside of the 2200 proper (e.g. the Communications Adaptor) and need be considered here.

These eight command lines are physically identical, and their functions are pre-assigned in the table below for the sake of consistancy between I/O devices.

```
ADR      EQU   1
STATUS   EQU   2
DATA     EQU   3
WRITE    EQU   4
COM1     EQU   5
COM2     EQU   6
COM3     EQU   7
COM4     EQU   8
```

(In all examples following in this chapter it is assumed that all External Command labels have been defined.)

When an External Command is executed, physically all that occurs in the processor is a pulse (or strobe) on the indicated command line. All other action occurs in one of the I/O devices.

a. EX ADR is the only command strobe acted upon by all I/O devices at the same time. All other command strobes affect only the I/O device that is currently addressed.

EXTERNAL COMMAND

COMMAND NUMBER	(exp)	OCTAL CODE	COMMAND	DESCRIPTION
1	ADR	121	Address	Selects device specified by A-register
2	STATUS	123	Sense Status	Connects selected device data lines to data input bus
3	DATA	125	Sense Data	Connects selected device data lines to data input bus
4	WRITE	127	Write Strobe	Signals selected device that output data is on data output lines
5	COM1	131	Command 1	Signals selected device that a control word is on data output lines
6	COM2	133	Command 2	Signals selected device that a control word is on data output lines
7	COM3	135	Command 3	Signals selected device that a control word is on data output lines
8	COM4	137	Command 4	Signals selected device that a control word is on data output lines

When external commands are to be used in a program the names or labels for the commands used should be defined to the assembler at the beginning of the source code listing as in the following example:

b. EX STATUS causes the selected device to place its status word on the input bus (it may already be on the bus in which case the EX STATUS does nothing).

c. EX DATA causes the selected device to place its data word on the input bus. This data will remain there until an EX STATUS or an EX ADR is executed.

d. EX WRITE- The write strobe command is a signal from the processor that data is present on the data output lines for the selected external device.

e. EX COM1 thru EX COM4 are used generally to load command words into I/O device command word registers. Depending on the device, however, they may be used for any purpose.

Device addressing in the Datapoint 2200 follows an unusual convention which the programmer should be aware. Up to 16 devices may be addressed, and the first four (low order) bits of the address word indicate which address is selected (zero through fifteen). The second four (high order) bits of the address word must contain the binary complement of the first four bits. Some of the sixteen possible addresses are reserved for specific devices. The remaining ones may be assigned as needed for a particular application.

DEVICE ADDRESS ASSIGNEMENTS

DEVICE	NUMBER	BINARY	OCTAL
Cassette Tape Decks	0	11110000	360
CRT/Keyboard	1	11100001	341
Communications Adaptor	2	11010010	322
2200P Printer	3	11000011	303
2200T Tape Transport	4	10110100	264
Unassigned	5	10100101	245
"	6	10010110	226
"	7	10000111	207
"	8	01111000	170
"	9	01101001	151
"	10	01011010	132
"	11	01001011	113
"	12	00111100	074
"	13	00101101	055
"	14	00011110	036
"	15	00001111	017

By way of example, to address (or select) the Communications Adaptor (and de-address all other devices) the following instructions are all that is required:

```
        ,
        ,
LA      0322
EX      ADR
        ,
        ,
        ,
```

3.3 The Input Command

In order to load the A-register with whatever is on the input bus an INPUT instruction is executed. In addition to loading the A-register with a new value, it transmits a strobe to the selected external device to inform it that the input bus has been read. Generally, if the status word is on the input bus, the input strobe is of no interest to the I/O device. However, if the data word from the I/O device is on the input bus, then the input strobe informs the I/O device that it has been read by the processor and the device then clears the read ready status bit.

3.4 Command Words

Through the use of the EX COM1 through EX COM4 strobes, it is possible to load command words in an I/O device, which causes the device to carry out specific instructions, or to assume some specific configuration. An excellent example of a command word structure is shown in paragraph 8.5 of the Datapoint 2200 Reference Manual. Each bit of the command word affects some aspect of the Communications Adaptor configuration and the entire operating mode of the adaptor is determined by the content of the Command Word Register at any given time.

When an EX COMn is executed, all of the bits in the affected command word register are loaded from the A-register so care must be taken that all eight bits are accounted for whenever a change is made in a command word register. Generally, when a word is loaded into a command word register, it remains there until another one replaces it. In some devices, (not the Communications Adaptor) a bit set to one will return to zero automatically when some function is carried out.

To give an example; suppose it was desired to instruct the Communications Adaptor to go "off-hook" and to "send 2025 Hz". The device would be addressed:

```
        ,
        ,
LA      0322
EX      ADR
```

and then a command word loaded

```
LA      060
EX      COM1
        ,
        ,
```

Where 060 is the octal value of the command word.

3.5 The Status Words

The status word provides a means of communicating to the processor the state of an I/O device at any given time. The status word is placed on the input bus whenever an I/O device is addressed, and remains there until the device is de-addressed or an EX DATA is executed. If the status of the device changes while it is selected, the value on the input bus changes with it, and may be read into the A-register without re-addressing the device. Paragraph 8.2 of the 2200 Reference Manual provides a detailed example of the status word structure used in the Communications Adaptor.

If it were desired to jump to a subroutine if the "Ringing Present" bit of this status word were to come true it could be coded as follows:

```
        ,
        ,
LA      0322
EX      ADR       ADDRESS DEVICE
INPUT             INPUT STATUS WORD
ND      040       MASK OFF ALL OTHER BITS
JFZ     SUBR      JUMP TO SUBROUTINE IF A CONTAINS A ONE
        ,
        ,
```

3.6 Character Buffers

An I/O device generally has one or more registers or buffers used to hold characters (also called "data") which are being transmitted or received by the device. Slow devices such as the keyboard usually have only one character buffer since the processor has plenty of time to read a character from the buffer before another is loaded.

Faster devices such as the Communications Adaptor have a double character buffer for transmitting or receiving data so that the processor may be reading (or writing) from one buffer while a data set (or some other external equipment) is writing (or reading) to the other buffer. This means that the processor always has at least one full character time in which to service the Communications Adaptor between data transfers.

Some even faster devices (such as the 2200T IBM Compatible Tape Deck) buffer an entire string of characters (up to 1024 in this example).

4. SIMPLE COMMUNICATIONS ADAPTOR ROUTINES

In writing any routine for the Communications Adaptor some simple rules must be followed. Reference should be made to Section 8 of the Datapoint 2200 Programmer's Manual in order to understand the following discussion.

Whenever data is to be transmitted or received through the Communications Adaptor, the device must first be configured for the mode of operation to be used. This is generally done with a prep subroutine which sets the Communications Adaptor Command Word (EX COM1). The transmit and receive time base registers (EX COM3 and EX COM2) and the Character Length Mask Word (EX COM4).

4.1 External Printers

Suppose it is desired to drive an external printer such as the Datapoint 3300P from the Communications Adaptor (The Datapoint 2200P connects directly to the I/O bus and does not use the Communications Adaptor). The 3300P is an EIA RS-232 interface serial printer, operates at 300 baud (bits/second) uses an 8-information-bit code, and works best with two stop units.

Referring to paragraph 8.5 of the Reference Manual we see that the Command Word can be all zeros. (No data set is involved and neither transmit or received data is inverted).

Referring to paragraph 8.6 we see that to transmit 300 baud the transmit time base must be loaded with 377 followed by 000. The receive time base need not be set since we are only transmitting to a printer.

Referring to paragraph 8.7 we see that the transmitted character length mask must be 001 (binary) and the receive character length can be 000 (binary) since we are not going to receive anything. Bit 6 must be 1 since we are using the EIA-RS-232 output. The binary value of this word then is 01000001 (binary) or 101 (octal).

The following subroutine will therefore configure the Communications Adaptor for the 3300P printer:

```
PREP1   LA      0322
        EX      ADR       ADDRESS DEVICE
        LA      0
        EX      COM1      SET COMMAND WORD
        LA      0377
        EX      COM3      SET TRANSMIT TIME BASE
        LA      0
        EX      COM3
        LA      0101
        EX      COM4      SET CHARACTER LENGTH
        RET
```

This routine need only be executed once at the start of the use of the printer.

Once the Communications Adaptor is configured a subroutine must be called to transmit data to the printer. An in-line subroutine could look like this:

```
PRINT1  HL      MSG       LOAD H AND L WITH BUFFER ADDRESS OF MESSAGE TO BE TRANSMITTED
LOOP    LA      0322
        EX      ADR
        INPUT             ADDRESS DEVICE
        ND      1         MASK FOR TRANSMIT READY
        JTZ     LOOP      LOOP BACK IF NOT READY
        LAM               LOAD A FROM MEMORY IF READY
        EX      WRITE     TRANSMIT TO COMMUNICATIONS ADAPTOR
        CP      015       COMPARE WITH END OF MESSAGE CHARACTER
        RTZ               RETURN IF END OF MESSAGE
        CALL    INCHL*    INCREMENT H AND L
        JMP     LOOP      LOOP BACK IF NOT END OF MESSAGE
```

The above example assumes that a message has been stored in a buffer area in memory and is transmitted to the printer to the exclusion of all other activity.

A more general routine might be to transmit a single character to the printer and the return to the calling program for other activity while the printer is printing. An example of this might be as follows:

```
PRINT   LA      0322
        EX      ADR       ADDRESS DEVICE
        INPUT
        ND      1         MASK FOR TRANSMIT READY
        RTZ               RETURN TO CALLING PROGRAM WITH A ZERO IN A REGISTER IF PRINTER
                          NOT READY
        LAB               IF PRINTER READY, LOAD A FROM B WITH CHARACTER TO BE PRINTED
        EX      WRITE
        OR      1         MAKE SURE Z-FLAG IS SET TO 0
        RET               RETURN TO CALLING PROGRAM
```

*See end of chapter for frequently used utility routines

Before calling this subroutine, B is loaded with the character to be printed and when the subroutine returns the Z-flag can be tested to see if the printer accepted the character.

4.2 Non-Automatic Data Sets

Data sets that are not automatically controlled from the software such as acoustic couplers or external data sets using private line connections are generally the easiest to program and will be used as our first examples of programming for data sets.

For an example, let us program a Datapoint 2200 to interface with an acoustic coupler which will be used to call a time-sharing service and operate full-duplex at 110 baud. (This program will make the 2200 look like a typical KSR teletype machine). The main program might be written like this:

```
START1  CALL   PREP2    CONFIGURE COMM ADAPTOR
        LA     012      (LINE FEED)

        CALL   DISPLY   CLEAR BOTTOM LINE OF CRT
SCAN1   CALL   READ1    INPUTS CHAR FROM COMM
                        ADAPTOR IF ONE READY
        JTZ    SCAN2    GO TO KYBD CHECK
HDX     CALL   DISPLY   WRITE CHAR IN A-REG TO CRT
SCAN2   CALL   KEYIN    INPUTS CHAR FROM KYBD
                        IF ONE READY
        JTZ    SCAN1    CHECK COMM ADAPTOR
        CALL   WRITE1   OUTPUT KYBD CHAR TO
                        COMM ADAPTOR
        JMP    SCAN1
```

This is all there needs to be to the main program. When starting, a prep subroutine is called to configure the Communications Adaptor. A scanning loop is then entered which looks for characters from the Communications Adaptor or the keyboard and transmits them to their respective destinations.

If it were desired to operate the program in a half-duplex mode where the characters are displayed directly on the CRT rather than full-duplex where the characters are transmitted back from the remote computer then the last instruction in the main program should be JMP HDX rather than JMP SCAN1.

In this particular mode of operation the Command Word would have bits 0 and 4 set to one and all others set to zero (Paragraph 8.5, 2200 Reference Manual). The time base mask words would be 375 and 106 for both transmit and receive, and the Character Length Mask word would be 111 (octal). (Bit 6 is set to one since the acoustic coupler is an external data set and uses the EIA-RS-232 interface.)

The PREP2 subroutines would therefore be coded as follows:

```
PREP2   LA    0322
        EX    ADR      ADDRESS DEVICE
        LA    021      OUTPUT COMMAND WORD
        EX    COM1     '
        LA    0375     SET TRANSMIT AND RECEIVE
        EX    COM2     TIME BASES TO 110 BAUD
        EX    COM3     '
        LA    0106     '
        EX    COM2     '
        EX    COM3     '
        LA    0111     SET CHAR LENGTH MASK

        EX    COM4     TO 11-UNIT CODE
        RET
```

To input characters from the Communications Adaptor a subroutine READ1 is written. It will test the Communications Adaptor to see if a character is ready, and if so, read it. If no character is found the Z-flag is returned set to 1 and if a character is read it is returned set to zero. The code is as follows:

```
READ1   LA    0322
        EX    ADR      ADDRESS DEVICE
        IN             CHECK READ READY AND
        ND    2        RETURN IF NOT READY
        RTZ
        EX    DATA     PUT DATA ON INPUT BUS
        IN             TRANSFER CHAR TO A-REG
        ORA            SET Z-FLAG IF CHAR = 0
        RET
```

To output characters to the Communications Adaptor a subroutine WRITE 1 is written. It will accept a character in the A-register, transmit it to the Communications Adaptor, and return to the main program when the task is finished with the character remaining in the A-register. It is coded as follows:

```
WRITE1  LBA   SAVE A IN B
        LA    0322
        EX    ADR      ADDRESS DEVICE
RETRY   IN             TEST FOR TRANSMIT READY
        ND    1        AND RETRY IF BUSY
        JTZ   RETRY
        LAB
        EX    WRITE    SEND CHAR OUT
        RET
```

The subroutines DISPLY and KEYIN are shown at the end of this chapter for information purposes. Since they do not involve the Communications Adaptor they will not be discussed here.

4.3 The High Level Keyer
When the high level keyer is used it operates in every respect like an external data set except that the Command Word is set to all zeros. Bit 6 of the Character Length Mask is set to one.

5. AUTOMATIC DATA SET OPERATION

One of the major features of the Datapoint 2200 is its ability to operate with the telephone network, providing completely automatic call origination and answering.

5.1 Automatic DDD Network Call Origination.
Automatic Call origination requires the Communications Adaptor to be provided with either a 103 or 202 internal data set option. These data sets interface with the telephone network through a Bell System Direct Access Arrangment (DAA). (See the Datapoint 2200 Installation Manual for specific details).

To automatically originate a call the following events must occur:

a. The DAA must have been on-hook long enough to assure complete termination of any previous call.

b. The Communications Adaptor must be configured for an automatic dialing mode.

c. The DAA must be set "Off-hook" and the dial tone present bit tested for ready (one).

d. The desired number transmitted.

e. The Communications Adaptor configured for the type of data set used and the connection confirmed (answered by another data set). (If the call is not confirmed within a reasonable time, usually about 30 seconds, a retry is probably indicated about 3 to 5 times).

f. Normal data transmission occurs.

g. The DAA is set to "on-hook" as soon as the connection is no longer desired.

The following code (page 8-9) provides an example of a complete automatic call origination sequence up to the point of reconfiguring the Communications Adaptor for the particular data set used (Step e. above). The number to be dialed is assumed to have been previously stored in an ASCII character sequence in a buffer area in memory beginning at NUMBER. An ASCII '*' (052) between digits results in an extra delay between dial pulses when such might be required to obtain an outside line in a private exchange or for some other reason. The end of the number is indicated by an ASCII return (015). For example:

```
NUMBER DC  '9*5125551234',015
```

would cause 9 to be dialed, then a pause, then 512-555-1234 to be dialed then control transferred to the calling program.

All other characters in the buffer area are ignored.

```
DIAL     HL     PHNUMB    BUFFER POINTER
         LA     0322
         EX     ADR       ADDRESS DEVICE
         SUA
         EX     COM1      SET DAA ON-HOOK
         DE     10000
         CALL   DELAY     DELAY 5 SECONDS
         LA     0330      CONFIGURE FOR DIALING; OFF-HOOK,
         EX     COM1      INVERT DATA, SEND DIAL PULSES.
         LA     0375      SET 100 BAUD (10 CPS) DIAL RATE
         EX     COM3
         LA     0
         EX     COM3
         EX     COM4
DTONE    IN               WAIT FOR DIAL TONE
         ND     0200      '
         JTZ    DTONE     '
         JMP    LDIG      GET FIRST DIGIT
NEXDIG   CALL   INCHL     INCREMENT H AND L
LDIG     LAM
         CP     '*'       IF THE A-REG CONTAINS '*' THEN
         JFZ    CMPR      CALL 5 SECOND DELAY
         DE     10000
         CALL   DELAY     '
         JMP    NEXDIG
CMPR     CP     015
         RTZ              RETURN IF END OF NUMBER
         CP     '9'+1     TEST FOR VALID DIGIT
         JFS    ERR1      '
         CP     '0'       '
         JTS    ERR1      '
         JFZ    MASK
         LA     10        CHANGE ZERO TO TEN
MASK     ND     017       MASK-OFF HIGH ORDER BITS
         LBA              SAVE A IN B
PLOOP    IN               WAIT FOR TRANSMIT READY
         ND     1
         JTZ    PLOOP     '
         LA     0360
         EX     WRITE     SEND DIAL PULSE
         LAB              DECREMENT PULSE COUNTER
         SU     1         '
         LBA
         DE     2000
         JFZ    PLOOP
         CALL   DELAY*    DELAY ONE SECOND
         JMP    NEXDIG
```

Upon returning from the DIAL subroutine the Communications Adaptor should be reconfigured for the type of data set used and the status bit tested for main channel carrier present. If it is not received within 30 seconds the call should be terminated and retried. The following code shows how this could be done for a 103 type data set operating at 150 baud.

---

*See end of chapter for frequently used utility routines

```
CNFIRM   LA      0322
         EX      ADR
         LA      0121
         EX      COM1
         LA      0376
         EX      COM2          RE-CONFIGURE COMMUNICATIONS ADAPTOR
         EX      COM3
         LA      0
         EX      COM2
         EX      COM3
         DE      60000         SET TIME COUNTER
MCCDET   INPUT
         ND      0100          RETURN IF MAIN CHANNEL CARRIER PRESENT
         RFZ
         LAE                   DECREMENT TIME COUNTER
         SU      1
         LEA
         LAD
         SB      0
         LDA
         JFZ     MCCDET
         RET                   RETURN IF TIME OUT
```

This subroutine returns with the Z-flag O if the carrier has been detected and 1 if it has not after 30 seconds.

The first part of a main program using these routines might be coded as follows:

```
START    LC      5             SET NUMBER OF CALL TRIES
RSTART   LAC
         SU      1             DECREMENT CALL COUNTER
         LCA
         JTZ     QUIT          EXIT AFTER 5 TRIES
         CALL    DIAL
         CALL    CONFIRM
         JTZ     RSTART
         ,
         ,
         ,                     REMAINDER OF MAIN PROGRAM
         ,
         ,
NUMBER   DC      '9*5125551234',015
```

5.2 Automatic DDD Network Call Answering.
Answering a call from the DDD network is very simple and we will not repeat coding examples for this function. The procedure is as follows:

a. Make sure the Command Word bit 4 is zero, maintaining the DAA "on-hook".

b. At regular intervals test bit 5 of the Status Word for ringing present.

c. If ringing is detected configure the Communications Adaptor for the type of data set used and set Command Word bit 5 for off-hook (1).

d. Depending on the type of data set, test for a received carrier (main channel in a type 103, main or supervisory channel depending on initial direction of communications in a 202 type). If no carrier is received after 30 seconds return to step a. above). If normal carrier is received then continue with normal communications.

6. FREQUENTLY USED SUBROUTINES

6.1 INCHL

This subroutine is used to increment the value stored in the H and L register as a double precision (16-bit) number.

```
INCHL   LAL
        AD      1
        LLA
LA      LAH
        AC      0
        LHA
        RET
```

6.2 DELAY

This subroutine provides a means for a time delay up to 30 seconds. Before calling the routine a double precision number is loaded into the D and E registers using the DE macro. This number is decremented at a rate of 2000 counts per second until D and E are zero and then the subroutine returns to the calling program.

```
DELAY   LAE
        SU      1
        LEA
        LAD
        SB      0
        LDA
        JFZ     DELAY
        ADE
        RTZ
        JMP     DELAY
```

6.3 DISPLY

This routine accepts a character in the A-register and displays it on the CRT screen at the current cursor position and then increments the cursor to the next position. Characters are always entered on the bottom line of the screen and the screen is rolled up one line whenever an ASCII line-feed is received (012). The character displayed is in the A-register when the routine returns.

```
,
,
,
DISPLY  LBA                 SAVE A IN B
        LA      0341        ADDRESS DEVICE
        EX      ADR
        LAB                 LOAD A FROM B AND
        ND      0177        MASK PARITY BIT
        CP      015         TEST FOR CR
        JTZ     CRDET
        CP      012         TEST FOR LF
        JTZ     LFDET
        CP      040         TEST FOR VALID
        RTS                 ASCII CHARACTER
        CP      0177        (RUBOUT)
        RTZ
        EX      WRITE
        HL      CURPOS      INCREMENT CURSOR POS
        LAM
        AD      1
        CP      80
        JFS     OFDET
        LMA                 SAVE CURSOR POS
        LCA                 SAVE A IN C
WCOMP   IN                  TEST FOR WRITE DONE
        ND      1
        JTZ     WCOMP
        HL      CMDWRD      GET COMMAND WORD
        LAM
        EX      COM1
        LA      020
        LMA                 RESTORE COMMAND WORD
        LAC
        EX      COM2        WRITE NEW CURSOR POS
        LA      11          MAINTAIN CURSOR ON
        EX      COM3        BOTTOM LINE
        LAB                 RESTORE CHAR TO A
        RET
LFDET   HL      CMDWRD
        LA      030         SET NEW
        LMA                 COMMAND WORD
        HL      CURPOS
        LCM                 LOAD CURSOR POS
        JMP     WCOMP
LFDET   HL      CMDWRD
        LA      030         SET NEW
        LMA     030         COMMAND WORD
        LC      0
        HL      CURPOS      SET NEW CURSOR
        LMC                 POSITION AND STORE
        JMP     WCOMP
CURPOS  DC      0
CMDWRD  DC      020
CRDET   EQU     LFDET
```

6.4 KEYIN

This subroutine is used to scan the keyboard and if a character is present return it to the calling program in the A-register. If the keyboard switch is held down during a keyboard entry, bit six of the data word is set to 0 allowing upper case ASCII characters to be converted to ASCII control characters (e.g., upper case J is converted to ASCII line-feed). The subroutines exits with the Z-flag set to one if no character is input and set to zero if a character is present.

```
KEYIN   LA    0341     ADDRESS DEVICE
        EX    ADR
        IN             INPUT STATUS
        LBA            SAVE STATUS IN B
        ND    2
        RTZ            RETURN IF READ NOT READY
        LAB            RESTORE STATUS
        ND    4        MASK FOR KYBD SENSE SW
        EX    DATA
        IN             READ DATA FROM KYBD
        JFZ   CCONT    JUMP IF KYBD SW SET
        ORA            RESET Z-FLAG
        RET
CCONT   ND    077      MASK BIT 6
        RET
```

SECTION 9

DATAPOINT 2200

OPERATING SYSTEM LISTING

```
                              .
                              . PARITY CHECK THE BOOTSTRAPED DATA
                              .
00000  066 050 056 000        CKLOAD: HL    PSTART
00004  036 000                        LD    $-$          INITIALIZE XOR CHECK
00006  046 000                        LE    $-$          INITIALIZE CIRCLE CHECK
00010  307                    CKLOOP: LAM                GET A BYTE
00011  320                            LCA                SAVE IT
00012  253                            XRD                ACCUMULATE THE XOR PARITY
00013  330                            LDA
00014  302                            LAC
00015  254                            XRE                ACCUMULATE THE CIRCLE PARITY
00016  012                            SRC
00017  340                            LEA
00020  306                            LAL                INCREMENT HL
00021  004 001                HALT:   AD    1
00023  360                            LLA
00024  305                            LAH
00025  014 000                        AC    0
00027  350                            LHA
00030  074 002                        CP    PEND>8       STOP WHEN PAST END
00032  110 010 000                    JFZ   CKLOOP
00035  306                            LAL
00036  074 000                        CP    PEND
00040  110 010 000                    JFZ   CKLOOP
00043  303                            LAD                CHECK THE PARITY ACCUMULATIONS
00044  264                            ORE
00045  110 022 000                    JFZ   HALT+1
00050  066 054 056 000        PSTART: HL    SCLOOP       CLEAR LOW CORE TO HALT SHORT LOADS
00054  306                    SCLOOP: LAL                DECREMENT MEMORY POINTER
00055  024 001                        SU    1
00057  360                            LLA
00060  373                            LMD                CLEAR THE LOCATION
00061  110 054 000                    JFZ   SCLOOP       GO UNTIL LOCATION ZERO CLEAR
                              .
                              . BOOTSTRAP LOADS THE ZEROTH FILE
                              .
00064  016 000                BOOT$*  LB    0            LOAD FILE ZERO
00066  106 100 000                    CALL  LOAD$
00071  100 075 000                    JFC   RUN$         EXECUTE IF LOAD WAS OKAY
00074  377                            HALT
00075  104 064 000            RUN$*   JMP   BOOT$        OVERSTORED WITH STARTING ADDRESS
                              .
                              . 2200 BINARY IMAGE FILE LOADER
                              . UPON ENTRY THE B REGISTER SHOULD CONTAIN
                              . THE DESIRED FILE NUMBER (POSITIVE)
                              . FILE LABEL RECORD FORMAT: 0201/0176/N/-N
                              . DATA RECORD FORMAT: 0303/074/XP/CP/H/L/-H/-L/DATA...
                              . THE 0303/074 INDICATES NUMERIC TYPE DATA
                              . H AND L DEFINE THE STARTING ADDRESS
                              . XP IS THE XOR PARITY AND CP IS THE CIRCULAR PARITY
                              . FOR THE CHARACTERS FOLLOWING THE CP
                              .
00100  006 360                LOAD$*  LA    0360         ADDRESS THE CASSETTE MECHANISM
```

```
ØØ102  121                         EX    ADR
ØØ103  1Ø6 322 ØØ1                 CALL  STOP      STOP ANY TAPE MOTION
ØØ106  155                         EX    DECK1     SELECT THE SYSTEM DECK
ØØ107  1Ø4 121 ØØØ                 JMP   LOAD
ØØ112  ØØ6 36Ø           LOAD2$*   LA    Ø36Ø      ADDRESS THE CASSETTE MECHANISM
ØØ114  121                         EX    ADR
ØØ115  1Ø6 322 ØØ1                 CALL  STOP      STOP ANY TAPE MOTION
ØØ12Ø  157                         EX    DECK2     SELECT THE DATA DECK
ØØ121  1Ø6 323 ØØ1       LOAD:     CALL  DWAIT     WAIT FOR DECK SELECTION
ØØ124  3Ø1                         LAB             THE REQUESTED FILE NUMBER MUST BE
ØØ125  26Ø                         ORA             POSITIVE
ØØ126  16Ø 27Ø ØØ1                 JTS   ARGH
ØØ131  Ø66 Ø77 Ø56 ØØØ             HL    RUN$+2    INITIALIZE THE STARTING LOCATION MSB
ØØ135  25Ø                         XRA             FOR 'NOTHING LOADED' FLAG
ØØ136  37Ø                         LMA
ØØ137  1Ø4 Ø12 ØØ1                 JMP   FSTART

                         .
                         . SEARCH FOR THE DESIRED FILE
                         .
ØØ142  1Ø6 36Ø ØØ1       FWAIT:    CALL  GETCH     WAIT FOR END OF RECORD
ØØ145  1ØØ 142 ØØØ                 JFC   FWAIT
ØØ15Ø  1Ø6 35Ø ØØ1       FNEXT:    CALL  RTINIT    INITIALIZE THE RE-TRY COUNT
ØØ153  Ø26 ØØ6           FREAD:    LC    6         WAIT FOR DATA OR LEADER
ØØ155  1Ø6 325 ØØ1                 CALL  TWAIT
ØØ16Ø  Ø44 ØØ2                     ND    2         QUIT IF LEADER
ØØ162  11Ø 27Ø ØØ1                 JFZ   ARGH
ØØ165  1Ø6 36Ø ØØ1                 CALL  GETCH     GET THE RECORD TYPE
ØØ17Ø  33Ø                         LDA             SAVE IT
ØØ171  1Ø6 36Ø ØØ1                 CALL  GETCH     GET THE RECORD TYPE COMPLEMENTED
ØØ174  Ø54 377                     XR    Ø377      UN-COMPLEMENT IT
ØØ176  273                         CPD             THE TWO MUST MATCH
ØØ177  11Ø 244 ØØØ                 JFZ   FSTOP
ØØ2Ø2  Ø74 3Ø3                     CP    Ø3Ø3      IGNORE NUMERIC RECORDS
ØØ2Ø4  15Ø 142 ØØØ                 JTZ   FWAIT
ØØ2Ø7  Ø74 347                     CP    Ø347      IGNORE SYMBOLIC RECORDS
ØØ211  15Ø 142 ØØØ                 JTZ   FWAIT
ØØ214  Ø74 2Ø1                     CP    Ø2Ø1      ELSE IT MUST BE AN EOF RECORD
ØØ216  11Ø 244 ØØØ                 JFZ   FSTOP
ØØ221  1Ø6 36Ø ØØ1                 CALL  GETCH     GET THE FILE NUMBER
ØØ224  33Ø                         LDA             SAVE IT
ØØ225  1Ø6 36Ø ØØ1                 CALL  GETCH     GET THE FILE NUMBER COMPLEMENTED
ØØ23Ø  Ø54 377                     XR    Ø377      UN-COMPLEMENT IT
ØØ232  273                         CPD             MAKE SURE THE TWO MATCH
ØØ233  11Ø 244 ØØØ                 JFZ   FSTOP
ØØ236  1Ø6 36Ø ØØ1                 CALL  GETCH     MAKE SURE THIS IS THE END OF THE RECORD
ØØ241  14Ø 262 ØØØ                 JTC   WCHWAY
ØØ244  1Ø6 322 ØØ1       FSTOP:    CALL  STOP      STOP THE TAPE
ØØ247  167                         EX    BSP       BACK UP OVER THE RECORD
ØØ25Ø  1Ø6 334 ØØ1                 CALL  DECRTC    DECREMENT THE RE-TRY COUNT
ØØ253  16Ø 27Ø ØØ1                 JTS   ARGH      QUIT IF TOO MANY RE-TRIES
ØØ256  171                         EX    SF        RE-INITIATE FORWARD MOTION
ØØ257  1Ø4 153 ØØØ                 JMP   FREAD
ØØ262  3Ø3               WCHWAY:   LAD             SEE IF WE ARE THERE YET
ØØ263  271                         CPB
```

```
00264  160 150 000            JTS   FNEXT     KEEP GOING IF NOT FAR ENOUGH
00267  150 037 001            JTZ   NXTREC    START LOADING IF THERE
00272  106 322 001            CALL  STOP      ELSE STOP THE TAPE
00275  173                    EX    SB        AND START SEARCHING BACKWARD
00276  106 350 001   BWAIT:   CALL  RTINIT    INITIATE THE RE-TRY COUNT
00301  026 006                LC    6         WAIT FOR DATA OR LEADER
00303  106 325 001            CALL  TWAIT
00306  044 002                ND    2         QUIT IF LEADER
00310  110 270 001            JFZ   ARGH
00313  365           BREAD:   LLH             PUSH THE CHAR ONTO THE STACK
00314  354                    LHE
00315  343                    LED
00316  330                    LDA
00317  106 360 001            CALL  GETCH     GET THE NEXT RECORD CHARACTER
00322  100 313 000            JFC   BREAD
00325  304                    LAE             GET THE RECORD TYPE COMPLEMENTED
00326  054 377                XR    0377      UN-COMPLEMENT IT
00330  273                    CPD             IT MUST MATCH THE TYPE
00331  110 021 001            JFZ   BSTOP
00334  074 303                CP    0303      IGNORE NUMERIC RECORDS
00336  150 276 000            JTZ   BWAIT
00341  074 347                CP    0347      IGNORE SYMBOLIC RECORDS
00343  150 276 000            JTZ   BWAIT
00346  074 201                CP    0201      ELSE IT MUST BE AN EOF RECORD
00350  110 021 001            JFZ   BSTOP
00353  306                    LAL             GET THE FILE NUMBER COMPLEMENTED
00354  054 377                XR    0377      UN-COMPLEMENT IT
00356  225                    SUH             MAKE SURE IT MATCHES THE FILE NUMBER
00357  110 021 001            JFZ   BSTOP
00362  340                    LEA             FLIP OVER THE FILE NUMBER
00363  026 010                LC    8
00365  305           FLIP:    LAH
00366  012                    SRC
00367  350                    LHA
00370  304                    LAE
00371  210                    ACA
00372  340                    LEA
00373  302                    LAC
00374  024 001                SU    1
00376  320                    LCA
00377  110 365 000            JFZ   FLIP
00402  304                    LAE             COMPARE IT TO THE DESIRED FILE NUMBER
00403  271                    CPB
00404  160 270 001            JTS   ARGH      IT AINT THERE
00407  110 276 000            JFZ   BWAIT     WE HAVEN'T GONE BACK FAR ENOUGH
00412  106 322 001   FSTART:  CALL  STOP      ELSE STOP THE TAPE
00415  171                    EX    SF        AND START GOING FORWARD AGAIN
00416  104 150 000            JMP   FNEXT
00421  106 322 001   BSTOP:   CALL  STOP      TRY THAT RECORD IN REVERSE AGAIN
00424  161                    EX    RBK
00425  106 334 001            CALL  DECRTC    DECREMENT THE RE-TRY COUNT
00430  160 270 001            JTS   ARGH      QUIT IF TOO MANY RE-TRIES
00433  173                    EX    SB        RE-INITIATE BACKWARD MOTION
00434  104 313 000            JMP   BREAD
```

```
                                   .
                                   . READ IN A DATA RECORD HEADER
                                   .
00437  106 350 001                 NXTREC: CALL  RTINIT      INITIALIZE THE RE-TRY COUNT
00442  026 020                     NXTWAT: LC    020         WAIT FOR IRG
00444  106 325 001                         CALL  TWAIT
00447  106 360 001                 NEXTRY: CALL  GETCH       GET THE RECORD TYPE
00452  140 047 001                         JTC   NEXTRY      WAIT FOR DATA
00455  330                                 LDA               SAVE THE RECORD TYPE
00456  106 360 001                         CALL  GETCH       GET THE RECORD TYPE COMPLEMENTED
00461  054 377                             XR    0377        UN-COMPLEMENT IT
00463  273                                 CPD               THE TWO MUST MATCH
00464  110 304 001                         JFZ   AGAIN
00467  074 347                             CP    0347        IGNORE SYBOLIC RECORDS
00471  150 042 001                         JTZ   NXTWAT
00474  074 303                             CP    0303        LOAD NUMERIC RECORDS
00476  150 130 001                         JTZ   NXTONE
00501  074 201                             CP    0201        QUIT ON EOF MARKER
00503  110 304 001                         JFZ   AGAIN
00506  106 322 001                         CALL  STOP        STOP THE TAPE
00511  167                                 EX    BSP         BACK UP TO THE END OF THE FILE
00512  106 323 001                         CALL  DWAIT
00515  066 077 056 000                     HL    RUN$+2      MAKE SURE SOMETHING WAS LOADED
00521  307                                 LAM
00522  260                                 ORA
00523  150 270 001                         JTZ   ARGH        ERROR EXIT IF NOT
00526  250                                 XRA               ELSE SET THE ZERO CONDITION
00527  007                                 RET               AND QUIT
00530  106 360 001                 NXTONE: CALL  GETCH       GET THE PARITY INITIALIZATION VALUES
00533  350                                 LHA               IN H (XP) AND L (CP)
00534  106 360 001                         CALL  GETCH
00537  360                                 LLA
00540  106 360 001                         CALL  GETCH       GET THE STARTING ADDRESS IN DE
00543  330                                 LDA
00544  106 360 001                         CALL  GETCH
00547  340                                 LEA
00550  106 360 001                         CALL  GETCH       GET IT AGAIN FOR A CHECK
00553  054 377                             XR    0377        IT IS COMPLEMENTED THIS TIME
00555  273                                 CPD
00556  110 304 001                         JFZ   AGAIN
00561  106 360 001                         CALL  GETCH
00564  140 304 001                         JTC   AGAIN       CATCH THE RECORD BEING OVER ALREADY
00567  054 377                             XR    0377        UN-COMPLEMENT
00571  274                                 CPE
00572  110 304 001                         JFZ   AGAIN
00575  306                                 LAL               SAVE THE PARITY ACCUMULATORS
00576  325                                 LCH
00577  066 076 056 000                     HL    RUN$+1      STORE THE STARTING ADDRESS IN RUN$ JUMP
00603  374                                 LME
00604  066 077                             LL    RUN$+2
00606  373                                 LMD
00607  353                                 LHD               SET STORAGE POINTER TO STARTING ADDRESS
00610  364                                 LLE
00611  332                                 LDC               RESTORE THE PARITY ACCUMULATORS
```

```
00612  255                       XRH                 ACCUMULATE IN THE STARTING ADDRESS
00613  012                       SRC
00614  256                       XRL
00615  012                       SRC
00616  255                       XRH
00617  012                       SRC
00620  256                       XRL
00621  012                       SRC
00622  340                       LEA
00623  305                       LAH
                         .
                         . LOAD A RECORD ACCUMULATING PARITY
                         .
00624  106 360 001       NXTBYT: CALL  GETCH           GET A BYTE OF DATA
00627  140 277 001               JTC   EOR             CATCH END OF RECORD
00632  320                       LCA                   ELSE SAVE IT
00633  253                       XRD                   ACCUMULATE THE PARITIES
00634  330                       LDA
00635  302                       LAC
00636  254                       XRE
00637  012                       SRC
00640  340                       LEA
00641  306                       LAL                   PREVENT LOADING INTO THE LOADER
00642  024 000                   SU    PEND
00644  305                       LAH
00645  034 002                   SB    PEND>8
00647  160 270 001               JTS   ARGH
00652  372                       LMC                   STORE THE DATA IF ADDRESS OKAY
00653  306                       LAL                   INCREMENT THE MEMORY ADDRESS
00654  004 001                   AD    1
00656  360                       LLA
00657  305                       LAH
00660  014 000                   AC    0
00662  044 037                   ND    037             DO MEMORY WRAP-AROUND
00664  350                       LHA
00665  104 224 001               JMP   NXTBYT          GET THE NEXT DATA BYTE
00670  106 322 001       ARGH:   CALL  STOP            STOP THE TAPE
00673  064 001                   OR    1               INDICATE ABORTIVE EXIT WITH CARRY TOGGLE
00675  012                       SRC
00676  007                       RET
00677  303               EOR:    LAD                   CHECK PARITY ACCUMULATIONS
00700  264                       ORE
00701  150 037 001               JTZ   NXTREC
00704  106 322 001       AGAIN:  CALL  STOP            TRY THAT RECORD AGAIN
00707  167                       EX    BSP
00710  106 334 001               CALL  DECRTC          DECREMENT THE RE-TRY COUNT
00713  160 270 001               JTS   ARGH            QUIT IF TOO MANY RE-TRIES
00716  171                       EX    SF              RE-INITIATE FORWARD MOTION
00717  104 047 001               JMP   NEXTRY          AND TRY THE RECORD AGAIN
                         .
                         . UTILITY ROUTINES
                         .
00722  177               STOP:   EX    TSTOP           STOP THE TAPE
00723  026 001           DWAIT:  LC    1               WAIT FOR DECK READY
```

```
00725  123                  TWAIT:  EX     STATUS
00726  101                  WAITL:  IN
00727  242                          NDC
00730  150 326 001                  JTZ    WAITL       WAIT FOR SPECIFIED STATUS
00733  007                          RET
                            .
00734  106 323 001          DECRTC: CALL   DWAIT       WAIT FOR I/O OPERATION
00737  066 377 056 001              HL     RTC         DECREMENT THE RE-TRY COUNT
00743  307                          LAM
00744  024 001                      SU     1
00746  370                          LMA
00747  007                          RET
                            .
00750  066 377 056 001      RTINIT: HL     RTC         INITIATE THE RE-TRY COUNT
00754  006 003                      LA     3           TO TRY FOUR TIMES
00756  370                          LMA
00757  007                          RET
                            .
00760  123                  GETCH:  EX     STATUS      GET A CHARACTER
00761  101                          IN
00762  044 024                      ND     024         WAIT FOR DATA OR IRG
00764  150 360 001                  JTZ    GETCH
00767  002                          SLC
00770  002                          SLC
00771  002                          SLC
00772  002                          SLC
00773  043                          RTC                END OF RECORD
00774  125                          EX     DATA        ELSE GET THE CHARACTER
00775  101                          IN
00776  007                          RET
                            .
                            . SYSTEM STORAGE
                            .
00777  000                  RTC:    DC     0           RE-TRY COUNT
01000                       PEND:   EQU    $           END OF LOADER LOCATION

DONE
```

```
                                   .
                                   . OPERATING SYSTEM COMMAND DECODER
                                   .
05000                                      SET    05000
05000  016 001                             LB     1          LOAD THE TAPE DIRECTORY
05002  106 100 000                         CALL   LOAD$
05005  100 017 012                         JFC    GOODL      IT LOADED OKAY
05010  066 001 056 013                     HL     BDCMSG     ELSE PRINT CAT UN-LOADABLE MSG
05014  104 046 012                         JMP    NOCAT
05017  006 341                     GOODL:  LA     0341       KEYBOARD SWITCH OVERRIDES AUTO-LOAD
05021  121                                 EX     ADR
05022  101                                 IN
05023  044 004                             ND     4
05025  110 042 012                         JFZ    OS$
05030  066 171 056 037                     HL     ALPFN      RUN ANY AUTO-LOAD PROGRAM
05034  307                                 LAM
05035  260                                 ORA
05036  310                                 LBA
05037  110 201 037                         JFZ    MAUTO$
05042  066 303 056 012             OS$*    HL     OSMSG      PRINT THE START-UP MESSAGE
05046  106 151 036                 NOCAT:  CALL   DSPLY$
05051  066 367 056 012             NXTCMD: HL     RDYMSG     PRINT 'READY'
05055  106 151 036                         CALL   DSPLY$
05060  066 151 056 015                     HL     CMDBUF     INPUT THE COMMAND
05064  046 013 036 000                     DE     11         POSITION THE CURSOR FOR ENTRY
05070  026 024                             LC     20         ONLY ACCEPT 20 CHARACTER
05072  106 000 036                         CALL   KEYIN$
05075  066 362 056 012                     HL     CRLF
05101  106 151 036                         CALL   DSPLY$     DO CRLF AFTER COMMAND ENTRY
05104  250                                 XRA               KEEP THE CURSOR OFF
05105  131                                 EX     COM1
05106  066 150 056 015                     HL     INPTR      INITIALIZE THE SCANNER POINTER
05112  006 151                             LA     CMDBUF
05114  370                                 LMA
05115  106 316 013                         CALL   GETSYM     GET THE COMMAND SYMBOL
05120  066 200 056 037                     HL     SYMBOL+6   CHECK THE TERMINATING CHARACTER
05124  307                                 LAM
05125  074 015                             CP     015        IT MUST BE AN ENTER
05127  150 144 012                         JTZ    FNDCMD
05132  074 055                             CP     '-'        A DASH
05134  150 144 012                         JTZ    FNDCMD
05137  074 040                             CP     ' '        OR A SPACE
05141  110 264 012                         JFZ    BADCMD
05144  066 175 056 037             FNDCMD: HL     SYMBOL+3   USE ONLY THE FIRST THREE CHARACTERS
05150  016 040                             LB     ' '
05152  026 003                             LC     3
05154  106 040 014                         CALL   BLKSET
05157  046 000 036 015                     DE     CMDLST     LOOK IT UP IN THE COMMAND LIST
05163  106 264 037                         CALL   LOOKUP
05166  306                                 LAL               POINT THE MEMORY POINTER TO THE
05167  044 370                             ND     0370       BRANCH ADDRESS
05171  004 006                             AD     6
05173  360                                 LLA
05174  347                                 LEM
```

```
05175  106 353 036              CALL  INCHL
05200  337                      LDM
05201  066 214 056 012          HL    CBI+1          PUT THE ADDRESS IN THE JUMP INSTRUCTION
05205  374                      LME
05206  066 215 056 012          HL    CBI+2
05212  373                      LMD
05213  106 264 012      CBI:    CALL  BADCMD
05216  104 051 012              JMP   NXTCMD
                        .
                        . ERROR MESSAGES
                        .
05221  066 073 056 013  NAMREQ: HL    NRQMSG
05225  104 270 012              JMP   BADSPL
05230  066 111 056 013  NONAME: HL    NONMSG
05234  104 270 012              JMP   BADSPL
05237  066 031 056 013  BADNAM: HL    BDNMSG
05243  104 270 012              JMP   BADSPL
05246  066 042 056 013  CATFUL: HL    CFLMSG
05252  104 270 012              JMP   BADSPL
05255  066 057 056 013  DUPNAM: HL    DUPMSG
05261  104 270 012              JMP   BADSPL
05264  066 217 056 013  BADCMD: HL    BCMSG
05270  036 000          BADSPL: LD    0
05272  046 013                  LE    11
05274  106 151 036              CALL  DSPLY$
05277  151                      EX    BEEP
05300  104 051 012              JMP   NXTCMD

05303  011 000 013 000  OSMSG:  DC    011.0.013.0.021.011.23.013.11
05314  103 117 115 120          DC    'COMPUTER TERMINAL OPERATING SYSTEM'.023.023.023.015
05362  011 000 013 013  CRLF:   DC    011.0.013.11.015
05367  011 000 013 013  RDYMSG: DC    011.0.013.11.'READY'.015
05401  011 000 013 013  BDCMSG: DC    011.0.013.11.022.'CATALOG UNLOADABLE'.015
05431  102 101 104 040  BDNMSG: DC    'BAD NAME'.015
05442  114 111 102 122  CFLMSG: DC    'LIBRARY FULL'.015
05457  116 101 115 105  DUPMSG: DC    'NAME IN USE'.015
05473  116 101 115 105  NRQMSG: DC    'NAME REQUIRED'.015
05511  116 117 040 123  NONMSG: DC    'NO SUCH NAME'.015
05526  101 125 124 117  NOAMSG: DC    'AUTO NOT SET'.015
05543  101 125 124 117  AUTMSG: DC    'AUTO SET TO '
05557  040 040 040 040  AUTENT: DC    '        '.015
05566  011 000 013 013  CBTMSG: DC    011.0.013.11.022.'FRONT TAPE SCRATCH?'.015
05617  127 110 101 124  BCMSG:  DC    'WHAT?'.015
                        .
05625  001              D1PKT:  DC    1              DECK ONE IS LOGICAL FILE ONE
05626  002 016                  DA    TFRBUF
05630  000                      DC    0
05631  002              D2PKT:  DC    2              DECK TWO IS LOGICAL FILE TWO
05632  002 016                  DA    TFRBUF
05634  000                      DC    0
05635  003              CATPAK: DC    3              CATALOG IS LOGICAL FILE THREE
05636  004 037                  DA    CATW
05640  166                      DC    ALPFN-CATW+1
05641  005              OBJPKT: DC    5              OBJECT FILE IS LOGICAL FILE FIVE
```

```
05642  002 016                  DA     TFRBUF
05644  000                      DC     0
                         .
                         . CALCULATE A PHYSICAL FILE NUMBER FROM CATALOG ADDRESS
                         .
05645  024 010           NCALC: SU     CAT
05647  012                      SRC
05650  012                      SRC
05651  012                      SRC
05652  004 002                  AD     2
05654  007                      RET
                         .
                         . SCAN OFF A NAME AND LOOK IT UP
                         .
05655  106 316 013       GETNAM* CALL  GETSYM        GET THE NAME
05660  074 015           GETNAN: CP    015
05662  110 237 012              JFZ    BADNAM        TERMINATING CHARACTER MUST BE AN 015
05665  066 172                  LL     SYMBOL        GET THE FIRST CHARACTER
05667  307                      LAM
05670  074 040                  CP     ' '
05672  150 221 012              JTZ    NAMREQ        THERE MUST BE A NAME
05675  046 010 036 037   GETNAX: DE    CAT           LOOK IT UP IN THE CATALOG
05701  106 264 037              CALL   LOCKUP
05704  306                      LAL
05705  044 007                  ND     7
05707  150 230 012              JTZ    NONAME        IT ISN'T THERE
05712  306                      LAL                  SET TABLE POINTER TO BEGINNING OF ENTRY
05713  044 370                  ND     0370
05715  007                      RET
                         .
                         . OPERATING SYSTEM LEXICAL SCANNING SUBROUTINES
                         .
05716  016 040           GETSYM: LB    ' '           BLANK THE SYMBOL STORAGE
05720  026 007                  LC     7
05722  066 172 056 037          HL     SYMBOL
05726  106 040 014              CALL   BLKSET
05731  026 172                  LC     SYMBOL        INITIALIZE THE SYMBOL STORAGE POINTER
05733  106 010 014              CALL   GETCH         GET THE FIRST CHARACTER
05736  074 101           GETLTR: CP    'A'           OR BETWEEN A AND Z
05740  160 002 014              JTS    GETERM
05743  074 133                  CP     'Z'+1
05745  120 002 014              JFS    GETERM
05750  056 037           GETNBR: LH    SYMBOL>8      STORE THE CHARACTER
05752  362                      LLC
05753  370                      LMA
05754  302                      LAC                  BUMP THE STORAGE INDEX
05755  074 200                  CP     SYMBOL+6      UNLESS IT IS AT THE END OF THE STORAGE
05757  014 000                  AC     0
05761  320                      LCA
05762  106 010 014              CALL   GETCH         GET THE NEXT CHARACTER
05765  074 060                  CP     '0'           CHECK IT'S RANGE BETWEEN 0 AND 9
05767  160 002 014              JTS    GETERM
05772  074 072                  CP     '9'+1
05774  160 350 013              JTS    GETNBR
```

```
05777  104 336 013                  JMP   GETLTR
06002  066 200 056 037     GETERM:  HL    SYMBOL+6        STORE THE TERMINATING CHARACTER
06006  370                          LMA
06007  007                          RET
                           .
                           . GET THE NEXT CHARACTER
                           .
06010  066 150 056 015     GETCH:   HL    INPTR           GET THE INPUT POINTER
06014  307                          LAM
06015  310                          LBA                   SAVE IT
06016  004 001                      AD    1               BUMP IT TO THE NEXT CHARACTER
06020  370                          LMA
06021  361                          LLB                   GET THE CHARACTER POINTED TO
06022  307                          LAM
06023  074 015                      CP    015             EXIT IF NOT CR
06025  013                          RFZ
06026  066 150                      LL    INPTR           ELSE DECREMENT THE CHARACTER POINTER
06030  307                          LAM
06031  024 001                      SU    1
06033  370                          LMA
06034  250                          XRA                   AND EXIT WITH ZERO CONDITION TRUE
06035  006 015                      LA    015             AND WITH A 015
06037  007                          RET
                           .
                           . SET A BLOCK OF CORE TO THE B REGISTER CONTENTS
                           . STARTING ADDRESS IN HL; NUMBER OF POSITIONS IN C
                           .
06040  371                 BLKSET:  LMB
06041  106 353 036                  CALL  INCHL
06044  302                          LAC
06045  024 001                      SU    1
06047  320                          LCA
06050  110 040 014                  JFZ   BLKSET
06053  007                          RET
                           .
                           . STORAGE
                           .
06400                               TP
06400  103 101 124 040     CMDLST:  DC    'CAT '          COMMAND LIST
06406  000 017                      DA    CATCMD
06410  116 101 115 040              DC    'NAM '
06416  127 017                      DA    NAMCMD
06420  122 125 116 040              DC    'RUN '
06426  267 022                      DA    RUNCMD
06430  111 116 040 040              DC    'IN  '
06436  263 017                      DA    INCMD
06440  117 125 124 040              DC    'OUT '
06446  036 020                      DA    OUTCMD
06450  104 105 114 040              DC    'DEL '
06456  147 021                      DA    DELCMD
06460  122 105 120 040              DC    'REP '
06466  341 020                      DA    REPCMD
06470  101 125 124 040              DC    'AUT '
06476  344 022                      DA    AUTCMD
```

```
06500  115 101 116 040                  DC      'MAN    '
06506  031 023                          DA      MANCMD
06510  120 122 105 040                  DC      'PRE    '
06516  073 023                          DA      PRECMD
06520  110 105 130 040                  DC      'HEX    '
06526  250 023                          DA      HEXCMD
06530  104 105 102 040                  DC      'DEB    '
06536  200 034                          DA      DEBUG$
06540  040 040 040 040                  DC      '       '
06546  264 012                          DA      BADCMD
06550  000                     INPTR:   DC      0               INPUT SCANNER INDEX
06551                          CMDBUF:  SKIP    22
06577  000                     CATPTR:  DC      0               LIBRARY CATALOG POINTER
06600  000                     CSCPTR:  DC      0               CATALOG SCREEN POINTER
06601  040 040 040 040         CATSPS:  DC      '        '.3    CATALOG NAME PRINT STRING
06612  000                     ENTSAV:  DC      0               CATALOG ENTRY ADDRESS STORAGE
06613  000                     PFNSEL:  DC      0               PHYSICAL FILE NUMBER SELECTED
06614  000                     PFNCTR:  DC      0               PHYSICAL FILE NUMBER COUNTER
07000                                   TP
07000  000 000                          DC      0.0             PARITY STORAGE FOR I/O ROUTINES
07002                          TFRBUF:  SKIP    254             I/O TRANSFER BUFFER
                               .
                               . LIST THE CATALOG
                               .
07400  066 177 056 015         CATCMD:  HL      CATPTR          INITIALIZE THE CATALOG POINTER
07404  006 010                          LA      CAT
07406  370                              LMA
07407  066 200                          LL      CSCPTR          INITIALIZE THE SCREEN POSITION
07411  250                              XRA
07412  370                              LMA
07413  066 177 056 015         CATLOP:  HL      CATPTR          GET THE ADDRESS OF THE NEXT CAT ENTRY
07417  367                              LLM
07420  056 037                          LH      CAT>8
07422  307                              LAM                     GET THE FIRST CHARACTER
07423  074 040                          CP      ' '
07425  150 117 017                      JTZ     CATEND          LISTING IS FINISHED IF IT IS A SPACE
07430  074 052                          CP      '*'
07432  150 117 017                      JTZ     CATEND          OR AN ASTERISK
07435  046 201 036 015                  DE      CATSPS          TRANSFER NAME INTO PRINT STRING
07441  026 006                          LC      6
07443  106 345 037                      CALL    BLKTFR
07446  066 200 056 015                  HL      CSCPTR          GET THE CURSOR POSITION
07452  337                              LDM
07453  303                              LAD                     SEE IF WE NEED TO GO TO A NEW LINE
07454  074 111                          CP      73
07456  160 070 017                      JTS     CATMOR
07461  066 362 056 012                  HL      CRLF            PUT OUT CR LF IF SO
07465  106 151 036                      CALL    DSPLY$
07470  046 013                 CATMOR:  LE      11              ALWAYS PRINT ON LINE 11
07472  066 201 056 015                  HL      CATSPS          PRINT THE NAME
07476  106 151 036                      CALL    DSPLY$
07501  066 200 056 015                  HL      CSCPTR          UPDATE THE CRUSOR POSITION
07505  373                              LMD
07506  066 177                          LL      CATPTR          UPDATE THE CATALOG ENTRY POSITION
```

```
07510  307                         LAM
07511  004 010                     AD     8
07513  370                         LMA
07514  104 013 017                 JMP    CATLOP      DO NEXT ENTRY
07517  066 362 056 012   CATEND:   HL     CRLF        MAKE ROOM FOR NEXT COMMAND
07523  106 151 036                 CALL   DSPLY$
07526  007                         RET
                         .
                         . CHANGE THE FILE NAME
                         .
07527  106 316 013       NAMCMD:   CALL   GETSYM      GET THE OLD NAME
07532  074 054                     CP     ','
07534  110 237 012                 JFZ    BADNAM      IT MUST BE TERMINATED BY A COMMA
07537  066 172                     LL     SYMBOL
07541  307                         LAM
07542  074 040                     CP     ' '
07544  150 221 012                 JTZ    NAMREQ      THERE MUST BE A NAME
07547  046 010 036 037             DE     CAT         LOOK IT UP
07553  106 264 037                 CALL   LOOKUP
07556  306                         LAL
07557  044 007                     ND     7
07561  150 230 012                 JTZ    NONAME      IT MUST BE IN CATALOG
07564  335                         LDH                SAVE THE CATALOG POINTER
07565  306                         LAL
07566  044 370                     ND     0370
07570  066 177 056 015             HL     CATPTR
07574  370                         LMA
07575  066 200                     LL     CSCPTR
07577  373                         LMD
07600  106 316 013                 CALL   GETSYM      GET THE NEW NAME
07603  074 015                     CP     015
07605  110 237 012                 JFZ    BADNAM      THE NEW NAME MUST BE TERMINATED BY 015
07610  066 172 056 037             HL     SYMBOL
07614  307                         LAM
07615  074 040                     CP     ' '
07617  150 221 012                 JTZ    NAMREQ      THERE MUST BE A NEW NAME
07622  046 010 036 037             DE     CAT         IT MUST NOT ALREADY BE IN THE CATALOG
07626  106 264 037                 CALL   LOOKUP
07631  306                         LAL
07632  044 007                     ND     7
07634  110 255 012                 JFZ    DUPNAM
07637  066 177 056 015             HL     CATPTR      RESTORE THE CATALOG POINTER
07643  347                         LEM
07644  066 200                     LL     CSCPTR
07646  337                         LDM
07647  066 172 056 037             HL     SYMBOL      TRANSFER THE SYMBOL INTO THE CATALOG
07653  026 006                     LC     6
07655  106 345 037                 CALL   BLKTFR
07660  104 054 023                 JMP    UPCAT       UPDATE THE CATALOG FILE
                         .
                         . BRING A NEW OBJECT FILE INTO THE SYSGEM
                         .
07663  106 300 017       INCMD:    CALL   INGET       DO THE PART COMMON WITH HEXCMD
07666  046 241 036 013             DE     OBJPKT      GET TO THE BEGINNING OF THE INPUT FILE
```

```
07672  106 022 030                      CALL   PBOF$
07675  104 076 021                      JMP    REPFIL
                             .
                             .
07700  106 316 013           INGET:     CALL   GETSYM          GET THE NAME SYMBOL
07703  074 015                          CP     015
07705  110 237 012                      JFZ    BADNAM          TERMINATING CHARACTER MUST BE 015
07710  066 172                          LL     SYMBOL          GET THE FIRST CHARACTER
07712  307                              LAM
07713  074 040                          CP     ' '
07715  150 221 012                      JTZ    NAMREQ          THERE MUST BE A NAME
07720  046 010 036 037                  DE     CAT             LOOK UP THE NAME IN THE CATALOG
07724  106 264 037           INEXT:     CALL   LOOKUP
07727  074 052                          CP     '*'
07731  150 246 012                      JTZ    CATFUL          CATALOG FULL IF FIRST CHARACTER IS *
07734  306                              LAL
07735  044 007                          ND     7
07737  110 255 012                      JFZ    DUPNAM          ENTRY MUST NOT BE IN THE TABLE
07742  335                              LDH                    PUT THE NEW NAME IN CATALOG
07743  306                              LAL                    BUMP MEMORY POINTER TO START OF ENTRY
07744  044 370                          ND     0370
07746  340                              LEA
07747  066 212 056 015                  HL     ENTSAV          SAVE THE CATALOG ADDRESS
07753  370                              LMA
07754  066 172 056 037                  HL     SYMBOL
07760  026 006                          LC     6
07762  106 345 037                      CALL   BLKTFR
07765  066 212 056 015                  HL     ENTSAV          CALCULATE THE SELECTED FILE NUMBER - 1
07771  307                              LAM
07772  106 245 013                      CALL   NCALC
07775  370                              LMA                    SAVE THE SELECTED FILE NUMBER
07776  024 001                          SU     1
10000  046 225 036 013                  DE     D1PKT           POSITION DECK ONE TO THAT FILE
10004  106 033 030                      CALL   CPFN$
10007  046 225 036 013                  DE     D1PKT
10013  106 022 030                      CALL   PBOF$
10016  046 225 036 013                  DE     D1PKT           GET TO THE END OF THAT FILE
10022  106 017 030                      CALL   PEOF$           SO READY TO APPEND THE NEW ONE
10025  066 212 056 015                  HL     ENTSAV          AFTER THE NEW FILE MARKER RECORD
10031  307                              LAM
10032  106 174 023                      CALL   D1FNW
10035  007                              RET
                             .
                             . OUTPUT AN ELEMENT
                             .
10036  106 316 013           OUTCMD:    CALL   GETSYM          GET THE ELEMENT NAME
10041  074 052                          CP     '*'             CHECK THE TERMINATING CHAR
10043  150 166 020                      JTZ    OUTALL          COPY WHOLE SYSTEM TAPE IF *
10046  074 044                          CP     '$'
10050  150 166 020                      JTZ    OUTALL          COPY ALL BUT OS AND CAT IF $
10053  106 260 013                      CALL   GETNAN          ELSE DO THE REST OF GETNAM
10056  106 245 013                      CALL   NCALC           CALCULATE THE PHYSICAL FILE NUMBER
10061  046 225 036 013                  DE     D1PKT           POSITION SYSTEM TAPE TO THAT FILE
10065  106 033 030                      CALL   CPFN$
```

```
1ØØ7Ø  Ø46 225 Ø36 Ø13             DE    D1PKT
1ØØ74  1Ø6 Ø22 Ø3Ø                 CALL  PBOF$
1ØØ77  1Ø6 Ø73 Ø23                 CALL  PRECMD       PREP THE DATA TAPE
1Ø1Ø2  Ø46 241 Ø36 Ø13             DE    OBJPKT       POSITION TO THE OUTPUT FILE
1Ø1Ø6  1Ø6 Ø22 Ø3Ø                 CALL  PBOF$
1Ø111  Ø46 225 Ø36 Ø13    OUTTFR:  DE    D1PKT        PUT OUT THE FILE
1Ø115  1Ø6 ØØØ Ø3Ø                 CALL  SNFR$        READ A RECORD FROM THE SYSTEM TAPE
1Ø12Ø  14Ø 145 Ø2Ø                 JTC   OUTEND       CATCH END OF FILE
1Ø123  3Ø6                         LAL                CALCULATE THE LENGTH
1Ø124  Ø24 ØØ2                     SU    TFRBUF
1Ø126  Ø66 244 Ø56 Ø13             HL    OBJPKT+3     PUT IT IN THE OUTPUT FILE LENGTH
1Ø132  37Ø                         LMA
1Ø133  Ø46 241 Ø36 Ø13             DE    OBJPKT       WRITE OUT THE RECORD
1Ø137  1Ø6 ØØ6 Ø3Ø                 CALL  SBFW$
1Ø142  1Ø4 111 Ø2Ø                 JMP   OUTTFR       DO THE NEXT RECORD
1Ø145  Ø46 231 Ø36 Ø13    OUTEND:  DE    D2PKT        PUT FILE MARKER 127 ON OUTPUT FILE
1Ø151  ØØ6 177                     LA    127
1Ø153  1Ø6 Ø33 Ø3Ø                 CALL  CPFN$
1Ø156  Ø46 231 Ø36 Ø13             DE    D2PKT
1Ø162  1Ø6 Ø44 Ø3Ø                 CALL  TFNW$
1Ø165  ØØ7                         RET
1Ø166  Ø66 172            OUTALL:  LL    SYMBOL       THERE MUST NOT HAVE BEEN A NAME
1Ø17Ø  3Ø7                         LAM
1Ø171  Ø74 Ø4Ø                     CP    ' '
1Ø173  11Ø 237 Ø12                 JFZ   BADNAM
1Ø176  Ø66 166 Ø56 Ø13             HL    CBTMSG       MAKE SURE THE FRONT TAPE IS SCRATCH
1Ø2Ø2  1Ø6 151 Ø36                 CALL  DSPLY$
1Ø2Ø5  151                         EX    BEEP
1Ø2Ø6  377                         HALT
1Ø2Ø7  ØØ6 36Ø                     LA    Ø36Ø         ADDRESS DECK 2
1Ø211  121                         EX    ADR
1Ø212  1Ø6 146 Ø24                 CALL  DWAIT
1Ø215  157                         EX    DECK2
1Ø216  1Ø6 146 Ø24                 CALL  DWAIT
1Ø221  175                         EX    REWND        REWIND THE TAPE
1Ø222  1Ø6 146 Ø24                 CALL  DWAIT
1Ø225  Ø66 ØØØ Ø56 Ø26             HL    BOOTS        WRITE THE BOOT BLOCK
1Ø231  Ø46 ØØØ Ø36 Ø3Ø             DE    BOOTE
1Ø235  1Ø6 213 Ø23                 CALL  WBLOK
1Ø24Ø  1Ø6 146 Ø24                 CALL  DWAIT
1Ø243  Ø66 2ØØ Ø56 Ø37             HL    SYMBOL+6     SEE IF THIS IS A FULL COPY
1Ø247  3Ø7                         LAM                OR JUST FILES 2 TO THE END
1Ø25Ø  Ø24 Ø44                     SU    '$'
1Ø252  15Ø 257 Ø2Ø                 JTZ   OUTSYS       START COPYING FROM FILE ZERO
1Ø255  ØØ6 ØØ2                     LA    2            START COPYING FROM FILE TWO
1Ø257  Ø46 225 Ø36 Ø13    OUTSYS:  DE    D1PKT
1Ø263  1Ø6 Ø33 Ø3Ø                 CALL  CPFN$
1Ø266  Ø46 225 Ø36 Ø13             DE    D1PKT
1Ø272  1Ø6 Ø22 Ø3Ø                 CALL  PBOF$
1Ø275  Ø66 214 Ø56 Ø15             HL    PFNCTR       COPY THE TAPE USING FIRST HALF OF UPDATE
1Ø3Ø1  ØØ6 377                     LA    -1
1Ø3Ø3  37Ø                         LMA                SET UP TO START WRITING FILE MARKERS AT Z
1Ø3Ø4  1Ø6 346 Ø21                 CALL  UPDATØ
1Ø3Ø7  ØØ6 177                     LA    127          TERMINATE THE DATA TAPE
```

```
10311  046 231 036 013              DE    D2PKT       WITH FILE MARKER 127
10315  106 033 030                  CALL  CPFN$
10320  046 231 036 013              DE    D2PKT
10324  106 044 030                  CALL  TFNW$
10327  106 146 024                  CALL  DWAIT
10332  175                          EX    REWND       REWIND DECK 2
10333  106 146 024                  CALL  DWAIT
10336  104 054 023                  JMP   UPCAT
                           .
                           . REPLACE THE NAMED FILE
                           .
10341  106 255 013         REPCMD:  CALL  GETNAM      GET THE FILE NAME
10344  066 212 056 015              HL    ENTSAV      SAVE THE CATALOG ENTRY ADDRESS
10350  004 010                      AD    8
10352  370                          LMA
10353  024 010                      SU    8
10355  106 245 013                  CALL  NCALC       CALCULATE THE PHYSICAL FILE NUMBER
10360  066 213 056 015              HL    PFNSEL      SAVE IT
10364  370                          LMA
10365  046 241 036 013              DE    OBJPKT      POSITION TO THE INPUT FILE
10371  106 022 030                  CALL  PBOF$
10374  066 212 056 015              HL    ENTSAV      SEE IF THIS IS THE LAST ENTRY IN THE CATA
10400  367                          LLM
10401  056 037                      LH    CAT>8
10403  307                          LAM
10404  074 040                      CP    ' '
10406  150 053 021                  JTZ   REPUP       DO SPECIAL UPDATE IF IT IS
10411  074 052                      CP    '*'
10413  150 053 021                  JTZ   REPUP
10416  046 241 036 013              DE    OBJPKT      POSITION TO THE END OF THE INPUT FILE
10422  106 017 030                  CALL  PEOF$
10425  066 213 056 015              HL    PFNSEL      PUT OUT A FILE MARKER AFTER IT
10431  307                          LAM
10432  046 231 036 013              DE    D2PKT
10436  106 033 030                  CALL  CPFN$
10441  046 231 036 013              DE    D2PKT
10445  106 044 030                  CALL  TFNW$
10450  104 066 022                  JMP   UPDATE      AND THEN DO THE NORMAL UPDATE
10453  066 213 056 015     REPUP:   HL    PFNSEL      GET SELECTED FILE NUMBER
10457  307                          LAM
10460  046 225 036 013              DE    D1PKT       POSITION SYSTEM TAPE TO THAT FILE
10464  106 033 030                  CALL  CPFN$
10467  046 225 036 013              DE    D1PKT
10473  106 022 030                  CALL  PBOF$
10476  046 241 036 013     REPFIL:  DE    OBJPKT      READ AN INPUT RECORD
10502  106 000 030                  CALL  SNFR$
10505  140 132 021                  JTC   REPEND      CATCH END OF FILE
10510  306                          LAL               CALCULATE THE LENGTH
10511  024 002                      SU    TFRBUF
10513  066 230 056 013              HL    D1PKT+3
10517  370                          LMA
10520  046 225 036 013              DE    D1PKT
10524  106 006 030                  CALL  SBFW$       WRITE THE RECORD
10527  104 076 021                  JMP   REPFIL      DO THE NEXT RECORD
```

```
10532  006 040               REPEND: LA    32           FOLLOW THE FILE BY FILE MARKERS
10534  106 174 023                   CALL  D1FNW        32 AND 127
10537  006 177                       LA    127
10541  106 174 023                   CALL  D1FNW
10544  104 054 023                   JMP   UPCAT        UPDATE THE CATALOG FILE

                             .
                             . DELETE A NAMED FILE
                             .
10547  106 255 013           DELCMD: CALL  GETNAM       GET THE NAMED FILE
10552  340                           LEA                SAVE IT
10553  066 212 056 015               HL    ENTSAV       SAVE THE CATALOG ENTRY ADDRESS
10557  370                           LMA
10560  106 245 013                   CALL  NCALC        CALCULATE THE PHYSICAL FILE NUMBER
10563  066 213 056 015               HL    PFNSEL       SAVE IT
10567  370                           LMA
10570  066 171 056 037               HL    ALPFN        KILL AUTO PTR IF IT IS POINTING
10574  227                           SUM                TO THE FILE TO BE DELETED
10575  110 204 021                   JFZ   DELDEC
10600  370                           LMA
10601  104 213 021                   JMP   DELAUT
10604  120 213 021           DELDEC: JFS   DELAUT       DELETED FILE AFTER AUTO-POINTED FILE
10607  307                           LAM                ELSE BUMP DOWN THE AUTO POINTER
10610  024 001                       SU    1            TO CORRESPOND TO CATALOG SHIFT
10612  370                           LMA
10613  304                   DELAUT: LAE                SEE IF AN ENTRY FOLLOWS
10614  004 010                       AD    8
10616  056 037                       LH    CAT>8
10620  360                           LLA
10621  307                           LAM
10622  074 040                       CP    ' '
10624  150 275 021                   JTZ   DELAST       TAKE SPECIAL ACTION IF NOT
10627  074 052                       CP    '*'
10631  150 275 021                   JTZ   DELAST
10634  026 010               DELMOV: LC    8            SHIFT DOWN THE CATALOG
10636  106 345 037                   CALL  BLKTFR
10641  307                           LAM
10642  074 040                       CP    ' '
10644  150 254 021                   JTZ   DELEND       DONE WHEN NO NEXT ENTRY
10647  074 052                       CP    '*'
10651  110 234 021                   JFZ   DELMOV       OR AT CATALOG STOP ENTRY
10654  364                   DELEND: LLE                CLEAR THE LAST ENTRY VACATED
10655  006 040                       LA    ' '          BY THE MOVE
10657  370                           LMA
10660  106 073 023                   CALL  PRECMD       PREP THE DATA TAPE
10663  046 241 036 013               DE    OBJPKT       POSITION FRONT DECK TO OBJECT FILE
10667  106 022 030                   CALL  PBOF$
10672  104 066 022                   JMP   UPDATE       ANN DO THE NORMAL UPDATE
10675  066 213 056 015       DELAST: HL    PFNSEL       SCROG THE LAST FILE
10701  307                           LAM                POSITION THE SYSTEM TAPE TO THE
10702  024 001                       SU    1            SELECTED FILE MINUS ONE
10704  046 225 036 013               DE    D1PKT
10710  106 033 030                   CALL  CPFN$
10713  046 225 036 013               DE    D1PKT
10717  106 022 030                   CALL  PBOF$
```

```
10722  046 225 036 013              DE    D1PKT        POSITION TO THE END OF THE FILE
10726  106 017 030                  CALL  PEOF$
10731  066 212 056 015              HL    ENTSAV       DELETE THE ENTRY FROM THE CATALOG
10735  367                          LLM
10736  056 037                      LH    CAT>8
10740  006 040                      LA
10742  370                          LMA
10743  104 132 021                  JMP   REPEND       TERMINATE TAPE AND UPDATE CATALOG
                          .
                          . UPDATE THE SYSTEM TAPE
                          .
10746  066 214 056 015    UPDAT0:   HL    PFNCTR       WRITE THE CURRENT PFN ON DECK TWO
10752  307                          LAM                INCREMENT THE CURRENT PFN
10753  004 001                      AD    1
10755  370                          LMA
10756  046 231 036 013              DE    D2PKT
10762  106 033 030                  CALL  CPFN$
10765  046 231 036 013              DE    D2PKT        WRITE IT ON DECK 2
10771  106 044 030                  CALL  TFNW$
10774  046 225 036 013    UPDAT1:   DE    D1PKT        READ A RECORD FROM DECK 1
11000  106 000 030                  CALL  SNFR$
11003  140 030 022                  JTC   UPDAT2       CATCH EOF
11006  306                          LAL                CALCULATE ITS LENGTH
11007  024 002                      SU    TFRBUF
11011  066 234 056 013              HL    D2PKT+3      AND PUT IT IN THE WRITE PACKET
11015  370                          LMA
11016  046 231 036 013              DE    D2PKT
11022  106 006 030                  CALL  SBFW$        WRITE THE RECORD INCLUDING PARITIES
11025  104 374 021                  JMP   UPDAT1       DO THE NEXT RECORD
11030  046 225 036 013    UPDAT2:   DE    D1PKT        READ FILE NUMBER FROM DECK 1
11034  106 041 030                  CALL  TFNR$
11037  302                          LAC
11040  074 040                      CP    32
11042  160 346 021                  JTS   UPDAT0       MORE TO GO IF LESS THAN 32
11045  006 040                      LA    32           ELSE PUT FILE MARKER 32 ON DECK 2
11047  046 231 036 013              DE    D2PKT
11053  106 033 030                  CALL  CPFN$
11056  046 231 036 013              DE    D2PKT
11062  106 044 030                  CALL  TFNW$
11065  007                          RET
                          .
                          .
11066  066 213 056 015    UPDATE:   HL    PFNSEL       GET THE SELECTED PHYSICAL FILE NUMBER
11072  307                          LAM
11073  066 214 056 015              HL    PFNCTR       INITIALIZE THE PFN COUNTER
11077  370                          LMA
11100  004 001                      AD    1
11102  046 225 036 013              DE    D1PKT        POSITION TO THE FILE AFTER THE ONE' SELECT
11106  106 033 030                  CALL  CPFN$
11111  046 225 036 013              DE    D1PKT
11115  106 022 030                  CALL  PBOF$
11120  106 374 021                  CALL  UPDAT1       COPY SYSTEM TAPE TO DATA TAPE
11123  046 241 036 013              DE    OBJPKT       POSITION DATA TAPE TO THE OBJECT FILE
11117  106 022 030                  CALL  PBOF$
```

```
11132  Ø66 213 Ø56 Ø15            HL    PFNSEL      RE-INITIALIZE THE FILE COUNTER
11136  3Ø7                        LAM
11137  Ø66 214 Ø56 Ø15            HL    PFNCTR
11143  37Ø                        LMA
11144  Ø46 225 Ø36 Ø13            DE    D1PKT       POSITION DECK 1 TO SELECTED FILE
1115Ø  1Ø6 Ø33 Ø3Ø                CALL  CPFN$
11153  Ø46 225 Ø36 Ø13            DE    D1PKT
11157  1Ø6 Ø22 Ø3Ø                CALL  PBOF$
11162  1Ø4 174 Ø22                JMP   UPDAT4
11165  Ø46 225 Ø36 Ø13   UPDAT3:  DE    D1PKT       WRITE A FILE NUMBER ON DECK 1
11171  1Ø6 Ø44 Ø3Ø                CALL  TFNW$
11174  Ø46 231 Ø36 Ø13   UPDAT4:  DE    D2PKT       READ A RECORD FROM DECK 2
112ØØ  1Ø6 ØØØ Ø3Ø                CALL  SNFR$
112Ø3  14Ø 23Ø Ø22                JTC   UPDAT6      CATCH EOF
112Ø6  3Ø6                        LAL               CALCULATE IT'S LENGTH
112Ø7  Ø24 ØØ2                    SU    TFRBUF
11211  Ø66 23Ø Ø56 Ø13            HL    D1PKT+3     PUT IT IN THE WRITE PACKET
11215  37Ø                        LMA
11216  Ø46 225 Ø36 Ø13            DE    D1PKT       WRITE THE FILE
11222  1Ø6 ØØ6 Ø3Ø                CALL  SBFW$       INCLUDING THE PARITY CHARACTERS
11225  1Ø4 174 Ø22                JMP   UPDAT4      DO THE NEXT RECORD
1123Ø  Ø66 214 Ø56 Ø15   UPDAT6:  HL    PFNCTR      INCREMENT THE CURRENT PFN COUNTER
11234  3Ø7                        LAM
11235  ØØ4 ØØ1                    AD    1
11237  37Ø                        LMA
1124Ø  Ø46 225 Ø36 Ø13            DE    D1PKT       CHANGE THE PACKET NUMBER
11244  1Ø6 Ø33 Ø3Ø                CALL  CPFN$
11247  Ø46 231 Ø36 Ø13            DE    D2PKT       READ THE NEXT FILE NUMBER FROM DECK 2
11253  1Ø6 Ø41 Ø3Ø                CALL  TFNR$
11256  3Ø2                        LAC
11257  Ø74 Ø4Ø                    CP    32
11261  16Ø 165 Ø22                JTS   UPDAT3      DO THE NEXT FILE IF IT IS LESS THAN 32
11264  1Ø4 132 Ø21                JMP   REPEND      ELSE TERMINATE TAPE AND UPDATE CATALOG
                         .
                         . LOAD AND EXECUTE A FILE
                         .
11267  1Ø6 316 Ø13       RUNCMD:  CALL  GETSYM      GET THE FILE NAME
11272  Ø74 Ø52                    CP    '*'         LOAD OBJECT FILE IF *
11274  15Ø 311 Ø22                JTZ   RUNOBJ
11277  1Ø6 26Ø Ø13                CALL  GETNAN      ELSE LOOK UP NAME
113Ø2  1Ø6 245 Ø13                CALL  NCALC       CALCULATE THE PFN
113Ø5  31Ø                        LBA               RUN IT
113Ø6  1Ø4 2Ø1 Ø37                JMP   MAUTO$
11311  Ø66 172           RUNOBJ:  LL    SYMBOL      MAKE SURE THERE
11313  3Ø7                        LAM               WAS NO NAME BESIDES *
11314  Ø74 Ø4Ø                    CP    ' '
11316  11Ø 237 Ø12                JFZ   BADNAM
11321  Ø46 241 Ø36 Ø13            DE    OBJPKT      POSITION THE FILE FOR THE LOADER
11325  1Ø6 Ø22 Ø3Ø                CALL  PBOF$
1133Ø  Ø46 231 Ø36 Ø13            DE    D2PKT
11334  1Ø6 Ø25 Ø3Ø                CALL  BSP$
11337  Ø16 ØØ1                    LB    1           RUN THE OBJECT FILE
11341  1Ø4 212 Ø37                JMP   MAUT2$      ON THE FRONT DECK
```

```
                                  .
                                  . SET THE AUTO-LOAD POINTER
                                  .
11344  066 171 056 037   AUTCMD: HL    ALPFN          GET THE POINTER
11350  307                       LAM
11351  260                       ORA
11352  110 377 022               JFZ   AUTDUP         ERROR IF ALREADY SET
11355  106 255 013               CALL  GETNAM         ELSE GET THE NAME
11360  024 010                   SU    CAT            CALCULATE THE FILE NUMBER
11362  012                       SRC
11363  012                       SRC
11364  012                       SRC
11365  004 002                   AD    2
11367  066 171 056 037           HL    ALPFN          AND SET THE POINTER
11373  370                       LMA
11374  104 054 023               JMP   UPCAT          AND UPDATE THE CATALOG FILE
11377  024 002           AUTDUP: SU    2              CALCULATE TABLE ADDRESS
11401  002                       SLC
11402  002                       SLC
11403  002                       SLC
11404  004 010                   AD    CAT
11406  360                       LLA
11407  056 037                   LH    CAT>8
11411  046 157 036 013           DE    AUTENT
11415  026 006                   LC    6
11417  106 345 037               CALL  BLKTFR         PUT TABLE ENTRY IN STRING
11422  066 143 056 013           HL    AUTMSG
11426  104 270 012               JMP   BADSPL         AND PRINT IT
                                  .
                                  . RESET THE AUTO-LOAD POINTER
                                  .
11431  066 171 056 037   MANCMD: HL    ALPFN
11435  307                       LAM
11436  260                       ORA
11437  066 126 056 013           HL    NOAMSG
11443  150 270 012               JTZ   BADSPL         AUTO IS NOT SET
11446  066 171 056 037           HL    ALPFN
11452  250                       XRA
11453  370                       LMA
                                  .
                                  . UPDATE THE CATALOG FILE
                                  .
11454  046 235 036 013   UPCAT:  DE    CATPAK
11460  106 022 030               CALL  PBOF$
11463  046 235 036 013           DE    CATPAK
11467  106 011 030               CALL  SNFW$
11472  007                       RET
                                  .
                                  . PREPARE A BLANK DATA TAPE
                                  .
11473  066 166 056 013   PRECMD: HL    CBTMSG         WAIT FOR BLANK TAPE
11477  106 151 036               CALL  DSPLY$
11502  151                       EX    BEEP
11503  377                       HALT
11504  046 231 036 013           DE    D2PKT          REWIND THE DATA TAPE
```

```
11510  106 Ø36 Ø3Ø               CALL  TRW$
11513  Ø46 231 Ø36 Ø13           DE    D2PKT          WRITE A FILE NUMBER Ø ON IT
11517  ØØ6 ØØØ                   LA    Ø
11521  106 Ø33 Ø3Ø               CALL  CPFN$
11524  Ø46 231 Ø36 Ø13           DE    D2PKT
11530  106 Ø44 Ø3Ø               CALL  TFNW$
11533  Ø46 231 Ø36 Ø13           DE    D2PKT          WRITE A FILE NUMBER 1 ON IT
11537  ØØ6 ØØ1                   LA    1
11541  106 Ø33 Ø3Ø               CALL  CPFN$
11544  Ø46 231 Ø36 Ø13           DE    D2PKT
11550  106 Ø44 Ø3Ø               CALL  TFNW$
11553  ØØ6 177                   LA    127
11555  Ø46 231 Ø36 Ø13           DE    D2PKT          WRITE A FILE NUMBER 127 ON IT
11561  106 Ø33 Ø3Ø               CALL  CPFN$
11564  Ø46 231 Ø36 Ø13           DE    D2PKT
11570  106 Ø44 Ø3Ø               CALL  TFNW$
11573  ØØ7                       RET
                         .
                         . WRITE A FILE MARKER ON DECK 1
                         .
11574  Ø46 225 Ø36 Ø13   D1FNW:  DE    D1PKT
11600  106 Ø33 Ø3Ø               CALL  CPFN$
11603  Ø46 225 Ø36 Ø13           DE    D1PKT
11607  106 Ø44 Ø3Ø               CALL  TFNW$
11612  ØØ7                       RET
                         .
                         . WRITE A BLOCK TO TAPE
                         .
11613  163               WBLOK:  EX    WBK            FIRE UP THE WRITE
11614  317               WNEXT:  LBM                  GET THE DATA CHARACTER
11615  123               WWAIT:  EX    STATUS         WAIT FOR WRITE READY
11616  101                       IN
11617  Ø44 Ø1Ø                   ND    Ø1Ø
11621  15Ø 215 Ø23               JTZ   WWAIT
11624  3Ø1                       LAB                  WRITE THE DATA CHARACTER
11625  127                       EX    WRITE
11626  3Ø6                       LAL                  BUMP THE MEMORY POINTER
11627  ØØ4 ØØ1                   AD    1
11631  36Ø                       LLA
11632  3Ø5                       LAH
11633  Ø14 ØØØ                   AC    Ø
11635  35Ø                       LHA
11636  273                       CPD                  SEE IF AT END OF BLOCK YET
11637  11Ø 214 Ø23               JFZ   WNEXT          NO CHANCE
11642  3Ø6                       LAL
11643  274                       CPE                  TRY LSB
11644  11Ø 214 Ø23               JFZ   WNEXT
11647  ØØ7                       RET                  ELSE WE ARE DONE
                         .
                         . PUT A TSB TAPE INTO THE LIBRARY
                         .
11650  1Ø6 3ØØ Ø17       HEXCMD: CALL  INGET          DO THE PART THAT IS LIKE INCMD
11653  Ø46 231 Ø36 Ø13           DE    D2PKT
11657  1Ø6 Ø36 Ø3Ø               CALL  TRW$
```

```
11662  106 157 024     HEXASR: CALL  HEXRBK       SEARCH FOR THE FIRST STARTING ADDRESS
11665  066 007                 LL    HEXBUF+1
11667  307                     LAM
11670  074 053                 CP    '+'          THE FIRST CHARACTER MUST BE A +
11672  110 262 023             JFZ   HEXASR

11675  066 010         HEXGAD: LL    HEXBUF+2     GET THE STARTING ADDRESS
11677  106 256 024             CALL  HEXCON
11702  140 123 024             JTC   HEXERR       IT MUST BE FOUR GOOD HEX CHARACTERS
11705  321                     LCB                SAVE MSB
11706  106 256 024             CALL  HEXCON
11711  140 123 024             JTC   HEXERR
11714  066 004                 LL    HEXADR       SAVE THE ADDRESS
11716  372                     LMC
11717  066 005                 LL    HEXADR+1
11721  371                     LMB

11722  106 157 024     HEXREC: CALL  HEXRBK       LOAD A RECORD
11725  066 006                 LL    HEXBUF       GET THE FIRST CHARACTER
11727  307                     LAM
11730  074 012                 CP    012          IT MUST BE A LINE FEED
11732  110 123 024             JFZ   HEXERR
11735  066 007                 LL    HEXBUF+1     GET THE SECOND CHARACTER
11737  307                     LAM
11740  074 052                 CP    '*'          IGNORE RECORD IF *
11742  150 322 023             JTZ   HEXREC
11745  074 053                 CP    '+'          GET ADDRESS IF +
11747  150 275 023             JTZ   HEXGAD
11752  074 043         E       CP    '##'         END OF FILE IF #
11754  150 132 021             JTZ   REPEND
11757  066 114                 LL    HEXWBP       CONVERT THE HEX IN HEXBUF
11761  347                     LEM                TO BINARY IN HEXWBF
11762  066 007                 LL    HEXBUF+1
11764  106 256 024     HEXCL:  CALL  HEXCON
11767  140 006 024             JTC   HEXEC        QUIT IF NON-HEX CHARACTER
11772  306                     LAL                SWAP E AND L
11773  364                     LLE
11774  340                     LEA
11775  371                     LMB                STORE BINARY NUMBER
11776  306                     LAL                INCREMENT AND SWAP L AND E
11777  004 001                 AD    1
12001  364                     LLE
12002  340                     LEA
12003  104 364 023             JMP   HEXCL        DO NEXT HEX PAIR
12006  307             HEXEC:  LAM                TERMINATING CHAR MUST BE 023
12007  074 023                 CP    023
12011  150 027 024             JTZ   HEXWRT
12014  074 053                 CP    '+'          UNLESS THIS BLOCK IS TO BE CONTINUED
12016  110 123 024             JFZ   HEXERR
12021  066 114                 LL    HEXWBP       IN WHICH CASE. JUST UPDATE
12023  374                     LME                THE WRITE BUFFER POINTER
12024  104 322 023             JMP   HEXREC
12027  066 114         HEXWRT: LL    HEXWBP       ELSE RESET THE WRITE BUFFER PTR
12031  036 123                 LD    HEXWBF+4
```

```
12033   373                        LMD
12034   066 004                    LL     HEXADR       PUT THE STARTING ADDRESS IN BUFFER
12036   307                        LAM
12037   066 117                    LL     HEXWBF
12041   370                        LMA
12042   054 377                    XR     0377
12044   066 121                    LL     HEXWBF+2
12046   370                        LMA
12047   066 005                    LL     HEXADR+1
12051   307                        LAM
12052   066 120                    LL     HEXWBF+1
12054   370                        LMA
12055   054 377                    XR     0377
12057   066 122                    LL     HEXWBF+3
12061   370                        LMA
12062   304                        LAE                 CALCULATE THE CORE BLOCK LENGTH
12063   024 123                    SU     HEXWBF+4
12065   340                        LEA
12066   066 005                    LL     HEXADR+1     UPDATE THE CORE ADDRESS
12070   307                        LAM
12071   204                        ADE
12072   370                        LMA
12073   066 004                    LL     HEXADR
12075   307                        LAM
12076   014 000                    AC     0
12100   370                        LMA
12101   304                        LAE                 CALCULATE THE WRITE BLOCK LENGTH
12102   004 004                    AD     4            COMPENSATE FOR HL GIVEN TWICE
12104   066 003 056 025            HL     HEXPKT+3     PUT THE LENGTH IN THE PACKET
12110   370                        LMA
12111   046 000 036 025            DE     HEXPKT       WRITE THE BUFFER
12115   106 011 030                CALL   SNFW$
12120   104 322 023                JMP    HEXREC       AND DO THE NEXT RECORD

12123   106 146 024        HEXERR: CALL   DWAIT        TRY THAT RECORD AGAIN
12126   167                        EX     BSP
12127   106 146 024                CALL   DWAIT
12132   006 341                    LA     0341         UNLESS KEYBOARD SWITCH DEPRESSED
12134   121                        EX     ADR
12135   101                        IN
12136   044 004                    ND     4
12140   150 322 023                JTZ    HEXREC
12143   104 132 021                JMP    REPEND

12146   026 001            DWAIT:  LC     1            DECK WAIT LOOP
12150   123                TWAIT:  EX     STATUS
12151   101                        IN
12152   242                        NDC
12153   150 150 024                JTZ    TWAIT
12156   007                        RET

12157   006 360            HEXRBK: LA     0360         MAKE SURE THE CASSETTE IS ADDRESSED
12161   121                        EX     ADR
12162   106 146 024                CALL   DWAIT        READ A BLOCK
```

```
12165  157                         EX    DECK2      FROM DECK 2
12166  106 146 024                 CALL  DWAIT
12171  066 006 056 025             HL    HEXBUF     INTO HEXBUF
12175  161                         EX    RBK
12176  026 024            HEXRNX:  LC    024        WAIT FOR IRG OR DATA
12200  106 150 024                 CALL  TWAIT
12203  044 020                     ND    020
12205  110 146 024                 JFZ   DWAIT      QUIT IF IRG
12210  125                         EX    DATA       ELSE PUT DATA INTO BUFFER
12211  101                         IN
12212  044 177                     ND    0177       STRIP THE PARITY
12214  370                         LMA
12215  306                         LAL              BUMP THE MEMORY POINTER
12216  004 001                     AD    1
12220  360                         LLA
12221  104 176 024                 JMP   HEXRNX

12224  024 060            HEXGET:  SU    '0'        CONVERT HEX TO 4-BIT BINARY
12226  160 252 024                 JTS   HEXCEN
12231  074 012                     CP    10
12233  160 250 024                 JTS   HEXLOW
12236  024 007                     SU    7
12240  160 252 024                 JTS   HEXCEN
12243  074 020                     CP    16
12245  120 252 024                 JFS   HEXCEN
12250  260                HEXLOW:  ORA              CLEAR THE CARRY TIGGLE
12251  007                         RET
12252  064 001            HEXCEN:  OR    1          SET THE CARRY TIGGLE
12254  012                         SRC
12255  007                         RET

12256  307                HEXCON:  LAM              GET THE FIRST CHARACTER
12257  106 224 024                 CALL  HEXGET     CONVERT IT TO BINARY
12262  043                         RTC              QUIT IF NOT HEX
12263  012                         SRC              PUT IT IN LEFT HALF OF BYTE
12264  012                         SRC
12265  012                         SRC
12266  012                         SRC
12267  310                         LBA              SAVE IT
12270  306                         LAL              BUMP THE MEMORY POINTER
12271  004 001                     AD    1
12273  360                         LLA
12274  307                         LAM              GET THE SECOND CHARACTER
12275  106 224 024                 CALL  HEXGET     CONVERT IT TO BINARY
12300  043                         RTC              QUIT IF NOT HEX
12301  261                         ORB              MERGE THE TWO HALVES
12302  310                         LBA              LEAVE RESULT IN B REGISTER
12303  306                         LAL              BUMP THE MEMORY POINTER AGAIN
12304  004 001                     AD    1
12306  360                         LLA
12307  007                         RET


12400                              TP
```

```
12400  001                  HEXPKT: DC    1              OUTPUT FILE IS LOGICAL FILE ONE
12401  117 025                      DA    HEXWBF         WRITE FROM WRITE BUFFER
12403  000                          DC    0
12404  000 000              HEXADR: DA    0              CURRENT CORE ADDRESS
12406                       HEXBUF: SKIP  70
12514  123                  HEXWBP: DC    HEXWBF+4       WRITE BUFFER POINTER
12515  000 000                      DC    0.0            ROOM FOR PARITY CHECKS
12517  000 000 000 000      HEXWBF: DC    0.0.0.0        ROOM FOR H AND L
12523                               SKIP  128            ROOM FOR THE DATA
12723                       HEXWBE: EQU   $
                            .
13000                               SET   013000         ROOM FOR THE BOOT BLOCK
13000                       BOOTS:  SKIP  01000
14000                       BOOTE:  EQU   $

DONE
```

```
14000                     SET    014000
                      .
                      . OPERATING SYSTEM ROUTINE ENTRY POINT TABLE
                      .
14000  104 052 030    SNFR$*  JMP    SNFRX
14003  104 230 030    SSFR$*  JMP    SSFRX
14006  104 376 030    SBFW$*  JMP    SBFWX
14011  104 005 031    SNFW$*  JMP    SNFWX
14014  104 072 031    SSFW$*  JMP    SSFWX
14017  104 366 031    PEOF$*  JMP    PEOFX
14022  104 375 031    PBOF$*  JMP    PBOFX
14025  104 004 032    BSP$*   JMP    BSPX
14030  104 016 032    CPDN$*  JMP    CPDNX
14033  104 030 032    CPFN$*  JMP    CPFNX
                      .
14036  104 075 034    TRW$*   JMP    TRWX
14041  104 114 034    TFNR$*  JMP    TFNRX
14044  104 127 034    TFNW$*  JMP    TFNWX
                      .
14047  104 104 032    ERR$*   JMP    ERRX
                      .
                      . SERIAL NUMERIC FILE READ
                      .
14052  106 355 031    SNFRX:  CALL   RTCI        INITIALIZE THE RE-TRY COUNT
14055  106 152 032    SNFRS:  CALL   GETPKT      GET THE PACKET PARAMETERS
14060  106 027 033            CALL   RBK$        START READING THE RECORD
14063  106 363 032            CALL   READ$       GET THE RECORD TYPE
14066  330                    LDA                SAVE IT
44067  106 363 032            CALL   READ$       GET THE RECORD TYPE COMPLEMENTED
14072  054 377                XR     0377        UN-COMPLEMENT IT
14074  273                    CPD                MAKE SURE THEY MATCH
14075  110 217 030            JFZ    SNFRR       TRY AGAIN IF THEY DON'T
14100  074 201                CP     0201        SEE IF IT IS A FILE MARKER
14102  150 062 032            JTZ    FEACT       QUIT IF IT IS
14105  074 347                CP     0347        SEE IF IT IS A SYMBOLIC RECORD
14107  150 074 032            JTZ    TEACT       TYPE ERROR IF IT IS
14112  074 303                CP     0303        MAKE SURE IT IS A NUMERIC RECORD
14114  110 217 030            JFZ    SNFRR
14117  106 363 032            CALL   READ$       GET THE PARITY CHECKS
14122  330                    LDA
14123  370                    LMA                STORE PARITY IN FIRST BYTE OF BUFFER
14124  306                    LAL
14125  004 001                AD     1
14127  360                    LLA
14130  305                    LAH
14131  014 000                AC     0
14133  350                    LHA
14134  106 363 032            CALL   READ$
14137  140 217 030            JTC    SNFRR       TRY AGAIN IF RECORD OVER ALREADY
14142  340                    LEA
14143  370                    LMA                STORE PARITY IN SECOND BYTE OF BUFFER
14144  306                    LAL
14145  004 001                AD     1
14147  360                    LLA
```

```
14150  305                       LAH
14151  Ø14 ØØØ                   AC     Ø
14153  35Ø                       LHA
14154  1Ø6 363 Ø32     SNFRL:    CALL   READ$       READ THE REST OF THE RECORD
14157  14Ø 2Ø5 Ø3Ø               JTC    SNFRE       QUIT IT AT END OF RECORD
14162  37Ø                       LMA                STORE THE BYTE OF DATA
14163  32Ø                       LCA                SAVE IT
14164  253                       XRD                ACCUMULATE THE PARITIES
14165  33Ø                       LDA
14166  3Ø2                       LAC
14167  254                       XRE
1417Ø  Ø12                       SRC
14171  34Ø                       LEA
14172  3Ø6                       LAL                BUMP THE MEMORY POINTER
14173  ØØ4 ØØ1                   AD     1
14175  36Ø                       LLA
14176  3Ø5                       LAH
14177  Ø14 ØØØ                   AC     Ø
142Ø1  35Ø                       LHA
142Ø2  1Ø4 154 Ø3Ø               JMP    SNFRL       DO THE NEXT BYTE
142Ø5  3Ø3             SNFRE:    LAD                CHECK THE PARITY TOTALS
142Ø6  264                       ORE
142Ø7  11Ø 217 Ø3Ø               JFZ    SNFRR       TRY AGAIN IF THEY ARENT BOTH ZERO
14212  1Ø6 Ø16 Ø33               CALL   WAIT$       ELSE WAIT FOR THE OPERATION TO BE COMPLET
14215  25Ø                       XRA                CLEAR THE CARRY TIGGLE
14216  ØØ7                       RET                AND RETURN
14217  1Ø6 324 Ø31     SNFRR:    CALL   DECRTC      BACK UP AND TRY AGAIN
14222  12Ø Ø55 Ø3Ø               JFS    SNFRS       UNLESS RTC IS NEGATIVE
14225  1Ø4 1Ø2 Ø32               JMP    PEACT       IN WHICH CASE. PARITY ERROR EXIT
                       .
                       . SERIAL SYMBOLIC FILE READ
                       .
1423Ø  1Ø6 355 Ø31     SSFRX:    CALL   RTCI        INITIALIZE THE RE-TRY COUNT
14233  1Ø6 152 Ø32     SSFRS:    CALL   GETPKT      GET PACKET PARAMETERS
14236  1Ø6 Ø27 Ø33               CALL   RBK$        START THE READ
14241  1Ø6 363 Ø32               CALL   READ$       GET THE RECORD TYPE
14244  33Ø                       LDA                SAVE IT
14245  1Ø6 363 Ø32               CALL   READ$       GET THE RECORD TYPE COMPLIMENTED
1425Ø  Ø54 377                   XR     Ø377        UN-COMPLEMENT IT
14252  273                       CPD                THEY MUST MATCH
14253  11Ø 365 Ø3Ø               JFZ    SSFRR
14256  Ø74 2Ø1                   CP     Ø2Ø1        QUIT IF IT IS AN EOF RECORD
1426Ø  15Ø Ø62 Ø32               JTZ    FEACT
14263  Ø74 3Ø3                   CP     Ø3Ø3        TYPE ERROR IF IT IS A NUMERIC RECORD
14265  15Ø Ø74 Ø32               JTZ    TEACT
1427Ø  Ø74 347                   CP     Ø347        MAKE SURE IT IS A SYMBOLIC RECORD
14272  11Ø 365 Ø3Ø               JFZ    SSFRR
14275  1Ø6 363 Ø32               CALL   READ$       INITIALIZE THE PARITY ACCUMULATORS
143ØØ  33Ø                       LDA
143Ø1  1Ø6 363 Ø32               CALL   READ$
143Ø4  34Ø                       LEA
143Ø5  14Ø 365 Ø3Ø               JTC    SSFRR       TRY AGAIN IF THE RECORD IS OVER ALREADY
1431Ø  1Ø6 363 Ø32     SSFRL:    CALL   READ$       READ THE REST OF THE RECORD
14313  14Ø 35Ø Ø3Ø               JTC    SSFRE       QUIT IF THE RECORD IS ENDED
```

```
14316  260                      ORA            CHECK THE VERTICAL PARITY
14317  130 365 030              JFP    SSFRR   TRY AGAIN IF IT IS FALSE
14322  320                      LCA            SAVE THE BYTE
14323  044 177                  ND     0177    STRIP THE VERTICAL PARITY
14325  370                      LMA            STORE THE BYTE
14326  302                      LAC            ACCUMULATE THE PARITIES
14327  253                      XRD
14330  330                      LDA
14331  302                      LAC
14332  254                      XRE
14333  012                      SRC
14334  340                      LEA
14335  306                      LAL            BUMP THE MEMORY POINTER
14336  004 001                  AD     1
14340  360                      LLA
14341  305                      LAH
14342  014 000                  AC     0
14344  350                      LHA
14345  104 310 030              JMP    SSFRL   DO THE NEXT CHARACTER
14350  006 015         SSFRE:   LA     015     TERMINATE STRING WITH AN 015
14352  370                      LMA
14353  303                      LAD            CHECK THE PARITY SUMS
14354  264                      ORE
14355  110 365 030              JFZ    SSFRR   TRY AGAIN IF BOTH ARENT ZERO
14360  106 016 033              CALL   WAIT$   ELSE WAIT FOR THE OPERATION TO COMPLETE
14363  250                      XRA            CLEAR THE CARRY TOGGLE
14364  007                      RET            AND RETURN
14365  106 324 031     SSFRR:   CALL   DECRTC  BACK UP AND TRY AGAIN
14370  120 233 030              JFS    SSFRS   UNLESS RTC IS NEGATIVE
14373  104 102 032              JMP    PEACT   IN WHICH CASE. PARITY ERROR EXIT
                       .
                       . SERIAL BLOCK FILE WRITE
                       .
14376  106 152 032     SBFWX:   CALL   GETPKT
14401  342                      LEC            PUT THE LENGTH IN THE E REGISTER
14402  104 026 031              JMP    SBFWE
                       .
                       . SERIAL NUMERIC FILE WRITE
                       .
14405  106 152 032     SNFWX:   CALL   GETPKT  GET THE PACKET PARAMETERS
14410  106 277 032              CALL   SAVHL   SAVE THE BUFFER STARTING ADDRESS
14413  036 000                  LD     0       INITIALIZE THE PARITY ACCUMULATORS
14415  046 000                  LE     0
14417  307             SNFWPG:  LAM            GENERATE THE PARITY TOTALS
14420  106 213 031              CALL   PARGEN
14423  110 017 031              JFZ    SNFWPG
14426  106 034 033     SBFWE:   CALL   WBK$    START UP THE WRITE
14431  036 303                  LD     0303    WRITE OUT RECORD TYPE NUMERIC
14433  106 002 033              CALL   WRITE$
14436  036 074                  LD     074     WRITE OUT ITS COMPLEMENT
14440  106 002 033              CALL   WRITE$
14443  337             SNFWL:   LDM            WRITE OUT THE REST OF THE RECORD
14444  106 002 033              CALL   WRITE$
14447  306                      LAL            BUMP THE MEMORY POINTER
```

```
14450  004 001                  AD     1
14452  360                      LLA
14453  305                      LAH
14454  014 000                  AC     0
14456  350                      LHA
14457  304                      LAE           DECREMENT THE BUFFER LENGTH COUNT
14460  024 001                  SU     1
14462  340                      LEA
14463  110 043 031              JFZ    SNFWL
14466  106 016 033              CALL   WAIT$         WAIT FOR THE OPERATION TO BE COMPLETE
14471  007                      RET
                       .
                       . SERIAL SYMBOLIC FILE WRITE
                       .
14472  106 152 032     SSFWX:   CALL   GETPKT        GET THE PACKET PARAMETERS
14475  106 277 032              CALL   SAVHL         SAVE THE START OF BUFFER ADDRESS
14500  036 000                  LD     0             INITIALIZE THE PARITY ACCUMULATORS
14502  046 000                  LE     0
14504  307             SSFWPG:  LAM                  GENERATE THE PARITY TOTALS
14505  074 015                  CP     015           CHECK FOR END OF BUFFER
14507  150 131 031              JTZ    SSFWPS
14512  260                      ORA                  GENERATE THE VERTICAL PARITY BIT
14513  170 120 031              JTP    SSFWPT
14516  054 200                  XR     0200
14520  370             SSFWPT:  LMA                  WRITE OUT CORRECTLY PARITIED CHAR
14521  026 002                  LC     2             FAKE OUT PARGEN LENGTH COUNTER
14523  106 213 031              CALL   PARGEN
14526  104 104 031              JMP    SSFWPG
14531  106 232 031     SSFWPS:  CALL   PARSTO
14534  106 034 033              CALL   WBK$          START UP THE WRITE
14537  036 347                  LD     0347          PUT OUT RECORD TYPE SYMBOLIC
14541  106 002 033              CALL   WRITE$
14544  036 030                  LD     030           PUT OUT THE TYPE COMPLEMENTED
14546  106 002 033              CALL   WRITE$
14551  046 002                  LE     2             DONT CHECK FOR 015 IN 1ST TWO PARITY BYTE
14553  304             SSFWL:   LAE
14554  024 001                  SU     1             DECREMENT FUDGE COUNTER
14556  340                      LEA
14557  307                      LAM                  GET CHARACTER FROM BUFFER
14560  120 174 031              JFS    SSFWW         EREG NOT NEG SO DONT CHECK FOR 015
14563  074 015                  CP     015           CHECK FOR END OF STRING
14565  110 174 031              JFZ    SSFWW         NOT END OF STRING SO WRITE IT OUT
14570  106 016 033              CALL   WAIT$         ITS A 015 SO END OF STRING
14573  007                      RET                  SO RETURN.
14574  330             SSFWW:   LDA                  WRITE THE BUFFERED CHARACTER
14575  106 002 033              CALL   WRITE$
14600  306                      LAL                  BUMP THE MEMORY POINTER
14601  004 001                  AD     1
14603  360                      LLA
14604  305                      LAH
14605  014 000                  AC     0
14607  350                      LHA
14610  104 153 031              JMP    SSFWL         DO THE NEXT CHARACTER
```

```
14613  310                  PARGEN: LBA              SAVE THE BYTE
14614  253                          XRD
14615  330                          LDA
14616  301                          LAB
14617  254                          XRE
14620  012                          SRC
14621  340                          LEA
14622  106 353 036                  CALL   INCHL
14625  302                          LAC              DECREMENT THE BUFFER LENGTH COUNT
14626  024 001                      SU     1
14630  320                          LCA
14631  013                          RFZ              DO NEXT BYTE IF NOT ZERO
14632  306                  PARSTO: LAL              CALCULATE NUMBER OF SHIFT MOD 8
14633  066 333 056 032              HL     HLSAV+1
14637  227                          SUM
14640  044 007                      ND     7
14642  320                          LCA
14643  302                  PSLOOP: LAC              SHIFT CIRCULATING PARITY BACK THAT MANY
14644  024 001                      SU     1
14646  320                          LCA
14647  160 260 031                  JTS    PSTORE
14652  304                          LAE
14653  002                          SLC
14654  340                          LEA
14655  104 243 031                  JMP    PSLOOP
14660  106 316 032          PSTORE: CALL   RESHL     STORE THE CIRC. PARITY
14663  106 364 036                  CALL   DECHL
14666  374                          LME
14667  106 364 036                  CALL   DECHL     STORE THE XOR PARITY
14672  373                          LMD
14673  066 254 056 032              HL     PKTADR    GET THE PACKET PARAMETERS AGAIN
14677  347                          LEM
14700  066 255 056 032              HL     PKTADR+1
14704  337                          LDM
14705  106 152 032                  CALL   GETPKT
14710  302                          LAC              INIT THE BUFFER LENGTH
14711  004 002                      AD     2         COMPENSATE FOR THE TWO PARITY ACCUMS
14713  340                          LEA              PUT LENGTH IN E-REGISTER
14714  106 364 036                  CALL   DECHL     BACK UP BUFFER POINTER TO PARITY ACCUMS
14717  106 364 036                  CALL   DECHL
14722  250                          XRA              RETURN WITH ZERO CONDITION TRUE
14723  007                          RET

                            .
                            . BACK UP AND DECREMENT THE RE-TRY COUNT
                            .
14724  106 041 033          DECRTC: CALL   BKSP$     BACK UP ONE RECORD
14727  106 016 033          DCCRTC: CALL   WAIT$     WAIT FOR IT
14732  066 365 056 031              HL     RTC       DECREMENT THF RE-TRY COUNT
14736  307                          LAM
14737  024 001                      SU     1
14741  370                          LMA
14742  066 254 056 032              HL     PKTADR    RESTORE THE PACKET ADDRESS
14746  347                          LEM
14747  066 255 056 032              HL     PKTADR+1
```

```
14753 337                          LDM
14754 007                          RET
                          .
                          . INITIATE THE RE-TRY COUNT
                          .
14755 066 365 056 031     RTCI:    HL     RTC             SET THE RE-TRY COUNT TO THREE
14761 006 003                      LA     3
14763 370                          LMA
14764 007                          RET
                          .
14765 000                 RTC:     DC     0               RE-TRY COUNT STORAGE
                          .
                          . POSITION TO THE END OF THE FILE
                          .
14766 106 152 032         PEOFX:   CALL   GETPKT
14771 106 310 033                  CALL   PEF$
14774 007                          RET
                          .
                          . POSITION TO THE BEGINNING OF THE FILE (AFTER FILE NUMBER RE
                          .
14775 106 152 032         PBOFX:   CALL   GETPKT
15000 106 056 033                  CALL   PBF$
15003 007                          RET
                          .
                          . BACKSPACE ONE RECORD
                          .
15004 106 152 032         BSPX:    CALL   GETPKT
15007 106 041 033                  CALL   BKSP$
15012 106 016 033                  CALL   WAIT$
15015 007                          RET
                          .
                          . CHANGE PHYSICAL DEVICE NUMBER
                          .
15016 320                 CPDNX:   LCA                    SAVE THE PFN
15017 353                          LHD
15020 364                          LLE
15021 046 257 036 032              DE     LFT             INIT THE LFT INDEX
15025 104 037 032                  JMP    CLFT            THE REST IF LIKE CPFN$
                          .
                          . CHANGE PHYSICAL FILE NUMBER
                          .
15030 320                 CPFNX:   LCA                    SAVE THE PFN
15031 353                          LHD
15032 364                          LLE
15033 046 260 036 032              DE     LFT+1           INIT THE LFT INDEX
15037 307                 CLFT:    LAM                    GET THE LOGICAL FILE NUMBER
15040 260                          ORA
15041 160 247 032                  JTS    GDFNER          CHECK IT'S RANGE
15044 074 010                      CP     8
15046 120 247 032                  JFS    GDFNER
15051 002                          SLC                    INDEX INTO THE LFT
15052 204                          ADE
15053 360                          LLA
15054 303                          LAD
```

```
15Ø55  Ø14 ØØØ                    AC      Ø
15Ø57  35Ø                        LHA
15Ø6Ø  372                        LMC                     CHANGE THE PFN
15Ø61  ØØ7                        RET
                          .
                          . END ACTION RETURN POINTS
                          .
15Ø62  1Ø6 Ø41 Ø33        FEACT:  CALL    BKSP$           BACK UP TO THE END OF FILE
15Ø65  1Ø6 Ø16 Ø33                CALL    WAIT$           WAIT FOR IT
15Ø7Ø  Ø64 ØØ1                    OR      1               SET THE CARRY TOGGLE
15Ø72  Ø12                        SRC
15Ø73  ØØ7                        RET
15Ø74  1Ø6 Ø16 Ø33        TEACT:  CALL    WAIT$           WAIT FOR RECORD TO FINISH
15Ø77  Ø64 ØØ1                    OR      1               TYPE ERROR RETURNS NON-ZERO
151Ø1  ØØ7                        RET
151Ø2  ØØ6 ØØ4            PEACT:  LA      4               INTERNAL ERROR D IF PARITY ERROR
                          .
                          . INTERNAL ERROR HANDLER
                          .
151Ø4  Ø66 15Ø Ø56 Ø32    ERRX:   HL      ERRS
1511Ø  ØØ4 1ØØ                    AD      'A'-1
15112  37Ø                        LMA
15113  Ø66 125 Ø56 Ø32            HL      ERRMSG
15117  1Ø6 151 Ø36                CALL    DSPLY$
15122  1Ø4 Ø64 ØØØ                JMP     BOOT$
15125  Ø11 ØØØ Ø13 Ø13    ERRMSG: DC      Ø11.Ø.Ø13.11.'INTERNAL ERROR
1515Ø  Ø4Ø Ø15            ERRS:   DC      ' '.Ø15
                          .
                          . GET THE DEVICE NUMBER IN THE B REGISTER
                          . THE PHYSICAL FILE NUMBER IF 'PFN'
                          . THE LENGTH IN THE C REGISTER
                          . THE BUFFER STARTING ADDRESS IN HL
                          .
15152  Ø66 254 Ø56 Ø32    GETPKT: HL      PKTADR          SAVE THE PACKET ADDRESS
15156  374                        LME
15157  Ø66 255 Ø56 Ø32            HL      PKTADR+1
15163  373                        LMD
15164  353                        LHD                     GET THE LOGICAL FILE NUMBER
15165  364                        LLE
15166  3Ø7                        LAM
15167  26Ø                        ORA                     CATCH LOGICAL FILE NUMBER OUT OF RANGE
1517Ø  16Ø 247 Ø32                JTS     GDFNER
15173  Ø74 Ø1Ø                    CP      8
15175  12Ø 247 Ø32                JFS     GDFNER
152ØØ  ØØ2                        SLC                     INDEX INTO THE LOGICAL FILE TABLE
152Ø1  ØØ4 257                    AD      LFT
152Ø3  36Ø                        LLA
152Ø4  ØØ6 Ø32                    LA      LFT>8
152Ø6  Ø14 ØØØ                    AC      Ø
1521Ø  35Ø                        LHA
15211  317                        LBM                     GET THE DEVICE NUMBER IN THE B REGISTER
15212  1Ø6 353 Ø36                CALL    INCHL
15215  327                        LCM                     GET THE PHYSICAL FILE NR IN THE C REG
15216  Ø66 256 Ø56 Ø32            HL      PFN             SAVE IT IN CORE
```

```
15222  372                          LMC
15223  353                          LHD             GET THE BUFFER STARTING ADDRESS
15224  364                          LLE
15225  106 353 036                  CALL   INCHL
15230  347                          LEM
15231  106 353 036                  CALL   INCHL
15234  337                          LDM
15235  106 353 036                  CALL   INCHL
15240  327                          LCM             GET THE LENGTH
15241  353                          LHD             PUT THE BSA IN HL
15242  364                          LLE
15243  106 334 032                  CALL   ADR$     SELECT THE PROPER PHYSICAL DEVICE
15246  007                          RET
15247  006 010          GDFNER:     LA     8        LOGICAL FILE NUMBER OUT OF RANGE NETS
15251  104 047 030                  JMP    ERR$     YOU AN INTERNAL ERROR NUMBER EIGHT
                        .
15254  000 000          PKTADR:     DA     0        CURRENT PACKET ADDRESS STORAGE
15256  000              PFN:        DC     0        CURRENT PHYSICAL FILE NUMBER STORAGE
                        .
                        . OPERATING SYSTEM LOGICAL FILE TABLE
                        .
15257  000 000          LFT:        DC     0.0      LF0 IS A NULL DEVICE
15261  001 000                      DC     1.0      LF1 FOR DECK 1
15263  002 000                      DC     2.0      LF2 FOR DECK 2
15265  001 001                      DC     1.1      LF3 IS CTOS CATALOG
15267  002 000                      DC     2.0      LF4 IS CTOS DATA SOURCE FILE
15271  002 001                      DC     2.1      LF5 IS CTOS DATA OBJECT FILE
15273  000 000                      DC     0.0
15275  001 040                      DC     1.32     LF7 IS ASM OBJECT SCRATCH FILE
                        .
                        . UTILITY ROUTINES
                        .
15277  305              SAVHL:      LAH
15300  316                          LBL
15301  066 332 056 032              HL     HLSAV
15305  370                          LMA
15306  066 333 056 032              HL     HLSAV+1
15312  371                          LMB
15313  361                          LLB
15314  350                          LHA
15315  007                          RET
                        .
15316  066 332 056 032  RESHL:      HL     HLSAV
15322  307                          LAM
15323  066 333 056 032              HL     HLSAV+1
15327  367                          LLM
15330  350                          LHA
15331  007                          RET
                        .
15332  000 000          HLSAV:      DA     0
                        .
                        . CASSETTE MECHANISM DRIVER
                        .
15334  006 360          ADR$:       LA     0360     ADDRESS THE CASSETTE MECHANISM
```

```
15336  121                       EX    ADR
15337  3Ø1                       LAB               SELECT THE PROPER DECK
1534Ø  Ø74 ØØ1                   CP    1
15342  15Ø 357 Ø32               JTZ   DEK1AD
15345  Ø74 ØØ2                   CP    2
15347  15Ø 361 Ø32               JTZ   DEK2AD
15352  ØØ6 ØØ1                   LA    1           BAD DEVICE NUMBER IS ERROR 'A'
15354  1Ø4 Ø47 Ø3Ø               JMP   ERR$
15357  155              DEK1AD:  EX    DECK1       SELECT DECK ONE
1536Ø  ØØ7                       RET
15361  157              DEK2AD:  EX    DECK2       SELECT DECK TWO
15362  ØØ7                       RET
                        .
                        . READ A CHARACTER INTO THE A REGISTER
                        .
15363                   READ$:
15363  123              DEKRED:  EX    STATUS      WAIT FOR IRG OR READ READY
15364  1Ø1                       IN
15365  Ø44 Ø24                   ND    Ø24
15367  15Ø 363 Ø32               JTZ   DEKRED
15372  ØØ2                       SLC
15373  ØØ2                       SLC
15374  ØØ2                       SLC
15375  ØØ2                       SLC
15376  Ø43                       RTC               QUIT IF INTER-RECORD GAP
15377  125                       EX    DATA        ELSE GET THE CHARACTER
154ØØ  1Ø1                       IN
154Ø1  ØØ7                       RET
                        .
                        . WRITE A CHARACTER FROM THE D REGISTER
                        .
154Ø2                   WRITE$:
154Ø2  123              DEKWRT:  EX    STATUS
154Ø3  1Ø1                       IN
154Ø4  Ø44 Ø11                   ND    Ø11
154Ø6  15Ø ØØ2 Ø33               JTZ   DEKWRT
15411  Ø12                       SRC
15412  Ø43                       RTC               ERROR IF DECK READY
15413  3Ø3                       LAD               WRITE THE DATA
15414  127                       EX    WRITE
15415  ØØ7                       RET
                        .
15416                   WAIT$:
15416  Ø26 ØØ1          DEKWAT:  LC    1           WAIT FOR DECK READY
1542Ø  123              WAIT:    EX    STATUS
15421  1Ø1                       IN
15422  242                       NDC
15423  15Ø Ø2Ø Ø33               JTZ   WAIT
15426  ØØ7                       RET
                        .
                        . FIRE UP BLOCK READ
                        .
15427                   RBK$:
15427  1Ø6 Ø16 Ø33      DEKRBK:  CALL  DEKWAT      WAIT FOR THE DECK TO BE READY
```

```
15432  161                   EX     RBK          FIRE UP THE READ BLOCK
15433  007                   RET
                     .
                     . FIRE UP BLOCK WRITE
                     .
15434                WBK$:
15434  106 016 033   DEKWBK: CALL   DEKWAT       WAIT FOR THE DECK TO BE READY
15437  163                   EX     WBK          FIRE UP BLOCK WRITE
15440  007                   RET
                     .
                     . BACKSPACE ONE RECORD
                     .
15441                BKSP$:
15441  106 016 033   DEKBSP: CALL   DEKWAT
15444  167                   EX     BSP
15445  007                   RET
                     .
                     . REWIND THE TAPE
                     .
15446                REWND$:
15446  106 016 033   DEKREW: CALL   DEKWAT
15451  175                   EX     REWND
15452  106 016 033           CALL   DEKWAT
15455  007                   RET
                     .
                     . POSITION TO THE BEGINNING OF THE FILE
                     .
15456                PBF$:
15456  106 016 033   DEKPBF: CALL   DEKWAT       WAIT FOR ANY PREVIOUS OPERATIONS
15461  066 256 056 032       HL     PFN          GET THE DESIRED FILE NUMBER
15465  317                   LBM
15466  173                   EX     SB           START SEARCHING BACKWARDS
15467  104 131 033           JMP    BWAIT
                     .
15472  106 333 033   BBACK:  CALL   DEKSTP       STOP THE TAPE
15475  106 355 033           CALL   DEKFNS       SEARCH FOR A FILE MARKER
15500  104 111 033           JMP    FSKIP
15503  106 355 031   FNEXT:  CALL   RTCI         INITIALIZE THE RE-TRY COUNT
15506  106 375 033           CALL   DEKFNN       SEARCH FOR NEXT FILE MARKER
15511  303           FSKIP:  LAD                 SEE IF WE ARE THERE YET
15512  271                   CPB
15513  160 103 033           JTS    FNEXT        STILL FURTHER TO GO
15516  150 274 033           JTZ    DEKTHE       WE ARE THERE
15521  106 333 033           CALL   DEKSTP       ELSE STOP THE TAPE
15524  006 007               LA     7            ERROR EXIT SEVEN
15526  104 047 030           JMP    ERR$
15531  106 355 031   BWAIT:  CALL   RTCI         INITIALIZE THE RE-TRY COUNT
15534  026 006               LC     6            WAIT FOR READ READY OR LEADER
15536  106 020 033           CALL   WAIT
15541  044 002               ND     2
15543  110 256 033           JFZ    BSTOP        CATCH LEADER
15546  365           BREAD:  LLH                 PUSH THE CHARACTER ONTO THE STACK
15547  354                   LHE
15550  343                   LED
```

```
15551  330                     LDA
15552  106 363 Ø32             CALL   DEKRED       GET THE NEXT RECORD CHARACTER
15555  100 146 Ø33             JFC    BREAD
1556Ø  3Ø4                     LAE                 GET THE SECOND RECORD CHARACTER
15561  Ø54 377                 XR     Ø377         UN-COMPLEMENT IT
15563  273                     CPD                 SEE IF IT MATCHES THE FIRST
15564  11Ø 256 Ø33             JFZ    BSTOP
15567  Ø74 3Ø3                 CP     Ø3Ø3         IGNORE NUMERIC RECORDS
15571  15Ø 131 Ø33             JTZ    BWAIT
15574  Ø74 347                 CP     Ø347         IGNORE SYMBOLIC RECORDS
15576  15Ø 131 Ø33             JTZ    BWAIT
156Ø1  Ø74 2Ø1                 CP     Ø2Ø1         ELSE IT MUST BE A FILE MARKER
156Ø3  11Ø 256 Ø33             JFZ    BSTOP
156Ø6  3Ø6                     LAL                 GET THE FILE NUMBER COMPLEMENTED
156Ø7  Ø54 377                 XR     Ø377
15611  275                     CPH                 IT MUST MATCH THE FILE NUMBER
15612  11Ø 256 Ø33             JFZ    BSTOP
15615  Ø46 ØØØ                 LE     Ø            FLIP OVER THE FILE NUMBER
15617  Ø36 Ø1Ø                 LD     8
15621  3Ø5             FLIP:   LAH
15622  Ø12                     SRC
15623  35Ø                     LHA
15624  3Ø4                     LAE
15625  21Ø                     ACA
15626  34Ø                     LEA
15627  3Ø3                     LAD
1563Ø  Ø24 ØØ1                 SU     1
15632  33Ø                     LDA
15633  11Ø 221 Ø33             JFZ    FLIP
15636  3Ø4                     LAE                 COMPARE IT TO THE DESIRED FILE NUMBER
15637  271                     CPB
1564Ø  16Ø Ø72 Ø33             JTS    BBACK        WE MUST GO IN THE OTHER DIRECTION
15643  11Ø 131 Ø33             JFZ    BWAIT        WE AREN'T BACK FAR ENOUGH
15646  1Ø6 333 Ø33             CALL   DEKSTP       ELSE STOP THE TAPE
15651  161                     EX     RBK          POSITION TO AFTER THE FILE LABEL
15652  1Ø6 Ø16 Ø33             CALL   DEKWAT       WAIT FOR IT
15655  ØØ7                     RET                 AND QUIT
15656  1Ø6 333 Ø33     BSTOP:  CALL   DEKSTP       STOP THE TAPE
15661  161                     EX     RBK          TRY THAT RECORD AGAIN
15662  1Ø6 341 Ø33             CALL   DCKRTC       DECREMENT THE RE-TRY COUNT
15665  16Ø Ø7Ø Ø34             JTS    DEKBAD       QUIT IT TOO MANY RE-TRIES
1567Ø  173                     EX     SB           RE-INITIATE BACKWARD MOTION
15671  1Ø4 146 Ø33             JMP    BREAD
15674  1Ø6 333 Ø33     DEKTHE: CALL   DEKSTP       STOP THE TAPE
15677  167                     EX     BSP          APPROACH THE GAP FROM FORWARD DIRECTION
157ØØ  1Ø6 Ø16 Ø33             CALL   DEKWAT
157Ø3  161                     EX     RBK
157Ø4  1Ø6 Ø16 Ø33             CALL   DEKWAT
157Ø7  ØØ7                     RET                 AND QUIT
                       .
                       . POSITION TO THE END OF THE FILE
                       .
1571Ø                  PEF$:
1571Ø  1Ø6 355 Ø33     DEKPEF: CALL   DEKFNS       SEARCH FOR THE NEXT FILE MARKER
```

```
15713  106 333 033                  CALL  DEKSTP        STOP THE TAPE
15716  167                          EX    BSP           POSITION IT TO AFTER THE LAST RECORD
15717  106 016 033                  CALL  DEKWAT        IN A FORWARD DIRECTION
15722  167                          EX    BSP
15723  106 016 033                  CALL  DEKWAT
15726  161                          EX    RBK
15727  106 016 033                  CALL  DEKWAT
15732  007                          RET
                          .
                          . STOP THE TAPE AND RE-SELECT THE PROPER DECK
                          .
15733  177                DEKSTP:   EX    TSTOP         STOP THE TAPE
15734  106 016 033                  CALL  DEKWAT        WAIT FOR IT TO STOP
15737  007                          RET
                          .
                          . BACK UP THE TAPE AND DECREMENT THE RE-TRY COUNT
                          .
15740  167                DEKRTC:   EX    BSP
15741  106 016 033        DCKRTC:   CALL  DEKWAT
15744  066 365 056 031              HL    RTC
15750  307                          LAM
15751  024 001                      SU    1
15753  370                          LMA
15754  007                          RET
                          .
                          . SEARCH FORWARD FOR A FILE MARKER
                          .
15755  106 355 031        DEKFNS:   CALL  RTCI          INITIATE THE RE-TRY COUNT
15760  106 016 033        DEKFNA:   CALL  DEKWAT        WAIT FOR THE DECK TO BE READY
15763  171                          EX    SF            START FORWARD MOTION
15764  104 375 033                  JMP   DEKFNN        INSPECT THE NEXT RECORD
15767  106 363 032        DEKFNW:   CALL  DEKRED        WAIT FOR BLOCK TO BE OVER
15772  100 367 033                  JFC   DEKFNW
15775  026 004            DEKFNN:   LC    4             WAIT FOR DATA
15777  106 020 033                  CALL  WAIT
16002  106 363 032                  CALL  DEKRED        GET THE RECORD TYPE
16005  330                          LDA                 SAVE THE CHARACTER
16006  106 363 032                  CALL  DEKRED        GET THE RECORD TYPE COMPLEMENTED
16011  054 377                      XR    0377          UN-COMPLEMENT IT
16013  273                          CPD                 THEY MUST MATCH
16014  110 057 034                  JFZ   DEKFNE
16017  074 303                      CP    0303          IGNORE NON-FILE MARKERS
16021  150 367 033                  JTZ   DEKFNW
16024  074 347                      CP    0347
16026  150 367 033                  JTZ   DEKFNW
16031  074 201                      CP    0201          ELSE IT MUST BE A FILE MARKER
16033  110 057 034                  JFZ   DEKFNE
16036  106 363 032                  CALL  DEKRED        GET THE FILE NUMBER
16041  330                          LDA                 SAVE IT
16042  106 363 032                  CALL  DEKRED        GET THE FILE NUMBER COMPLEMENTED
16045  054 377                      XR    0377          UN-COMPLEMENT IT
16047  273                          CPD                 THEY MUST MATCH
16050  110 057 034                  JFZ   DEKFNE
16053  106 363 032                  CALL  DEKRED        THIS MUST BE THE END OF THE RECORD
```

```
16056  043                         RTC
16057  106 333 033         DEKFNE: CALL  DEKSTP     STOP THE TAPE
16062  106 340 033                 CALL  DEKRTC     BACK UP AND COUNT TRY
16065  120 360 033                 JFS   DEKFNA     TRY AGAIN IF NOT ALREADY TOO MANY
16070  006 002             DEKBAD: LA    2          ELSE UNLOADABLE RECORD
16072  104 047 030                 JMP   ERR$
                           .
                           . SPECIAL TAPE ROUTINES
                           .
16075  106 152 032         TRWX:   CALL  GETPKT     REWIND THE TAPE
16100  301                         LAB
16101  074 002                     CP    2
16103  006 001                     LA    1
16105  110 047 030                 JFZ   ERR$       ONLY REWIND THE FRONT DECK
16110  106 046 033                 CALL  REWND$
16113  007                         RET
                           .
16114  106 152 032         TFNRX:  CALL  GETPKT     READ A FILE NUMBER
16117  106 355 033         TRWFNR: CALL  DEKFNS     FIND A FILE MARKER
16122  106 333 033                 CALL  DEKSTP     STOP THE TAPE AFTER IT
16125  323                         LCD              PUT THE FILE NUMBER IN THE C REGISTER
16126  007                         RET
                           .
16127  106 152 032         TFNWX:  CALL  GETPKT     WRITE A FILE NUMBER
16132  106 016 033                 CALL  DEKWAT
16135  163                         EX    WBK        FIRE UP A WRITE
16136  036 201                     LD    0201       WRITE OUT THE FILE MARKER
16140  106 002 033                 CALL  DEKWRT
16143  036 176                     LD    0176       WRITE OUT ITS COMPLEMENT
16145  106 002 033                 CALL  DEKWRT
16150  066 256 056 032             HL    PFN
16154  337                         LDM              WRITE OUT THE FILE NUMBER
16155  106 002 033                 CALL  DEKWRT
16160  303                         LAD              WRITE OUT ITS COMPLEMENT
16161  054 377                     XR    0377
16163  330                         LDA
16164  106 002 033                 CALL  DEKWRT
16167  106 016 033                 CALL  DEKWAT     TERMINATE THE WRITE OPERATION
16172  007                         RET
```

DONE

```
16200                               SET    Ø16200
                                  .
                                  . ITSEY BITSEY DEBUG
                                  .
16200  Ø66 273 Ø56 Ø35    DEBUG$*  HL     CURADR
16204  347                         LEM
16205  Ø66 274                     LL     CURADR+1
16207  337                         LDM
16210  Ø26 ØØ5                     LC     5
16212  Ø66 3Ø6                     LL     DSPADR+4
16214  1Ø6 226 Ø35                 CALL   CONBIN
16217  Ø66 273                     LL     CURADR
16221  347                         LEM
16222  Ø66 274                     LL     CURADR+1
16224  357                         LHM
16225  364                         LLE
16226  347                         LEM
16227  Ø36 ØØØ                     LD     Ø
16231  Ø26 ØØ3                     LC     3
16233  Ø66 313 Ø56 Ø35             HL     DSPDAT+2
16237  1Ø6 226 Ø35                 CALL   CONBIN
16242  Ø66 275                     LL     DISP
16244  1Ø6 151 Ø36                 CALL   DSPLY$
16247  Ø66 315                     LL     INBUF
16251  Ø26 Ø1Ø                     LC     8
16253  1Ø6 ØØØ Ø36                 CALL   KEYIN$
16256  Ø66 315                     LL     INBUF
1626Ø  1Ø6 155 Ø35                 CALL   CONOCT
16263  Ø74 Ø15                     CP     Ø15
16265  152 355 Ø34                 CTZ    NEWADR
1627Ø  Ø74 111                     CP     'I'
16272  152 364 Ø34                 CTZ    INCADR
16275  Ø74 1Ø4                     CP     'D'
16277  152 ØØ2 Ø35                 CTZ    DECADR
163Ø2  Ø74 115                     CP     'M'
163Ø4  152 Ø2Ø Ø35                 CTZ    MODIFY
163Ø7  Ø74 Ø56                     CP     '.'
16311  152 Ø31 Ø35                 CTZ    ENTER
16314  Ø74 114                     CP     'L'
16316  152 121 Ø35                 CTZ    LSAVE
16321  Ø74 11Ø                     CP     'H'
16323  152 144 Ø35                 CTZ    HSAVE
16326  Ø74 1Ø7                     CP     'G'
1633Ø  152 111 Ø35                 CTZ    GET
16333  Ø74 1Ø6                     CP     'F'
16335  152 1Ø1 Ø35                 CTZ    FRONT
1634Ø  Ø74 117                     CP     'O'
16342  15Ø Ø42 Ø12                 JTZ    OS$
16345  Ø74 1Ø3                     CP     'C'
16347  15Ø Ø42 Ø35                 JTZ    GOTO
16352  1Ø4 2ØØ Ø34                 JMP    DEBUG$
                                  .
16355  Ø66 273            NEWADR:  LL     CURADR
16357  374                         LME
```

```
16360  066 274                    LL    CURADR+1
16362  373                        LMD
16363  007                        RET
16364  066 273          INCADR:   LL    CURADR
16366  307                        LAM
16367  004 001                    AD    1
16371  370                        LMA
16372  066 274                    LL    CURADR+1
16374  307                        LAM
16375  014 000                    AC    0
16377  370                        LMA
16400  250                        XRA
16401  007                        RET
16402  066 273          DECADR:   LL    CURADR
16404  307                        LAM
16405  024 001                    SU    1
16407  370                        LMA
16410  066 274                    LL    CURADR+1
16412  307                        LAM
16413  034 000                    SB    0
16415  370                        LMA
16416  250                        XRA
16417  007                        RET
16420  066 273          MODIFY:   LL    CURADR
16422  337                        LDM
16423  066 274                    LL    CURADR+1
16425  357                        LHM
16426  363                        LLD
16427  374                        LME
16430  007                        RET
16431  106 020 035      ENTER:    CALL  MODIFY
16434  056 035                    LH    CURADR>8
16436  106 364 034                CALL  INCADR
16441  007                        RET
16442  066 051          GOTO:     LL    BRANCH+1
16444  374                        LME
16445  066 052                    LL    BRANCH+2
16447  373                        LMD
16450  106 000 000      BRANCH:   CALL  0
16453  300              LSAVI:    LAA
16454  300              HSAVI:    LAA
16455  066 370 056 035            HL    ASAVE
16461  370                        LMA
16462  066 371                    LL    BSAVE
16464  371                        LMB
16465  066 372                    LL    CSAVE
16467  372                        LMC
16470  066 373                    LL    DSAVE
16472  373                        LMD
16473  066 374                    LL    ESAVE
16475  374                        LME
16476  104 200 034                JMP   DEBUG$

16501  016 001          FRONT:    LB    1
```

```
16503  106 112 000              CALL  LOAD2$
16506  104 115 035              JMP   GETLOD

16511  314             GET:     LBE
16512  106 100 000              CALL  LOAD$
16515  053             GETLOD:  RTZ
16516  151                      EX    BEEP
16517  250                      XRA
16520  007                      RET

16521  066 053 056 035 LSAVE:   HL    LSAVI
16525  016 306                  LB    0306
16527  304             HLSAVM:  LAE
16530  044 007                  ND    7
16532  002                      SLC
16533  002                      SLC
16534  002                      SLC
16535  074 070                  CP    070
16537  023                      RFS
16540  261                      ORB
16541  370                      LMA
16542  250                      XRA
16543  007                      RET

16544  066 054 056 035 HSAVE:   HL    HSAVI
16550  016 305                  LB    0305
16552  104 127 035              JMP   HLSAVM

                       .
                       . CONVERT OCTAL TO BINARY
                       .
16555  036 000         CONOCT:  LD    0
16557  343                      LED
16560  317             CONLOP:  LBM
16561  106 353 036              CALL  INCHL
16564  301                      LAB
16565  074 070                  CP    '8'
16567  023                      RFS
16570  074 060                  CP    '0'
16572  063                      RTS
16573  044 007                  ND    7
16575  320                      LCA
16576  303                      LAD
16577  044 037                  ND    037
16601  002                      SLC
16602  002                      SLC
16603  002                      SLC
16604  330                      LDA
16605  304                      LAE
16606  002                      SLC
16607  002                      SLC
16610  002                      SLC
16611  340                      LEA
16612  044 007                  ND    7
16614  263                      ORD
```

```
16615  330                       LDA
16616  304                       LAE
16617  044 370                   ND     0370
16621  262                       ORC
16622  340                       LEA
16623  104 160 035               JMP    CONLOP
                         .
                         . CONVERT BINARY TO OCTAL (RIGHT TO LEFT)
                         .
16626  304               CONBIN: LAE
16627  044 007                   ND     7
16631  004 060                   AD     '0'
16633  370                       LMA
16634  106 364 036               CALL   DECHL
16637  304                       LAE
16640  012                       SRC
16641  012                       SRC
16642  012                       SRC
16643  044 037                   ND     037
16645  340                       LEA
16646  303                       LAD
16647  012                       SRC
16650  012                       SRC
16651  012                       SRC
16652  330                       LDA
16653  044 340                   ND     0340
16655  264                       ORE
16656  340                       LEA
16657  303                       LAD
16660  044 037                   ND     037
16662  330                       LDA
16663  302                       LAC
16664  024 001                   SU     1
16666  320                       LCA
16667  110 226 035               JFZ    CONBIN
16672  007                       RET
                         .
                         . STORAGE
                         .
16673  000 000           CURADR: DA     0
16675  011 000 013 013   DISP:   DC     011.0.013.11.021
16702  040 040 040 040   DSPADR: DC     '
16711  040 040 040 015   DSPDAT: DC     '     .015
16715  040 040 040 040   INBUF:  DC     '

16770                            SET    016770
16770  000               ASAVE:  DC     0
16771  000               BSAVE:  DC     0
16772  000               CSAVE:  DC     0
16773  000               DSAVE:  DC     0
16774  000               ESAVE:  DC     0
16775  001 002                   DC     1.2

DONE
```

```
17000                        SET   017000
                       .
                       . KEYBOARD ENTRY ROUTINE
                       .
                       . ACCEPTS A STRING OF CHARACTERS FROM THE KEYBOARD AND PUTS
                       . THEM IN MEMORY STARTING WITH THE ADDRESS GIVEN IN THE H
                       . AND L REGISTERS AND AT A DISPLAY POSITION DESCRIBED BY THE
                       . D (HORZ) AND E (VERT) REGISTERS. THE MAXIMUM NUMBER OF
                       . CHARACTERS ACCEPTED IS TAKEN FROM THE C REGISTER UPON ENTRY
                       . OVERFLOW OFF THE END OF A DISPLAY LINE IS NOT PERMITTED
                       . AND IF EITHER THE MAXIMUM COUNT OR DISPLAY BOUNDARY IS
                       . EXCEEDED. SUCCESSIVE CHARACTERS WILL GO IN THE LAST
                       . POSITION OVER AND OVER. AN 015 WILL TERMINATE INPUT REQUEST
                       . THE CURSOR IS TURNED ON UPON ENTRY AND OFF UPON EXIT.
                       .
17000  006 341         KEYIN$* LA    0341      ADDRESS THE KEYBOARD
17002  121                     EX    ADR
17003  312                     LBC             LOAD THE MAX COUNT INTO THE CURRENT COUNT
17004  006 020                 LA    020       TURN ON THE CURSOR
17006  131                     EX    COM1
17007  106 326 036     KILOOP: CALL  CWAIT     MAKE SURE THE DISPLAY IS READY
17012  123             KWLOOP: EX    STATUS    GET A CHARACTER FROM THE KEYBOARD
17013  101                     IN
17014  044 002                 ND    2
17016  150 012 036             JTZ   KWLOOP
17021  125                     EX    DATA
17022  101                     IN
17023  074 010                 CP    010       CATCH BACKSPACE
17025  150 105 036             JTZ   KBSP
17030  074 030                 CP    030       CATCH DELETE
17032  150 113 036             JTZ   KDEL
17035  074 100                 CP    0100      REVERSE THE SHIFT KEY FUNCTION
17037  160 044 036             JTS   KSTORE
17042  054 040                 XR    040
17044  370             KSTORE: LMA             STORE THE CHARACTER
17045  074 015                 CP    015       CATCH THE ENTER KEY
17047  150 102 036             JTZ   KEND
17052  127                     EX    WRITE     ELSE DISPLAY THE CHARACTER
17053  303                     LAD             CATCH CURSOR AT SCREEN BOUNDARY
17054  074 117                 CP    79
17056  120 007 036             JFS   KILOOP
17061  301                     LAB             DECREMENT THE CHARACTER COUNT
17062  024 001                 SU    1
17064  160 007 036             JTS   KILOOP    ALREADY ABOVE THE MAXIMUM
17067  310                     LBA
17070  303                     LAD             BUMP THE CURSOR POSITION FOR REAL
17071  004 001                 AD    1
17073  330                     LDA
17074  106 353 036             CALL  INCHL     BUMP THE MEMORY LOACTION
17077  104 007 036             JMP   KILOOP    DO THE NEXT CHARACTER

17102  250             KEND:   XRA             TURN OFF THE CURSOR
17103  131                     EX    COM1
17104  007                     RET
```

```
17105  106 124 036   KBSP:    CALL  KBSPR      BACKSPACE ONE CHARACTER
17110  104 007 036            JMP   KILOOP
17113  106 124 036   KDEL:    CALL  KBSPR      BACKSPACE TO THE BEGINNING OF THE ENTRY
17116  110 113 036            JFZ   KDEL
17121  104 007 036            JMP   KILOOP
                     .
17124  301           KBSPR:   LAB              INCREMENT THE CHARACTER COUNTER
17125  272                    CPC              UNLESS AT THE BEGINNING OF THE ENTRY
17126  053                    RTZ
17127  004 001                AD    1
17131  310                    LBA
17132  303                    LAD              DECREMENT THE SCREEN POSITION
17133  024 001                SU    1
17135  330                    LDA
17136  106 364 036            CALL  DECHL      DECREMENT THE MEMORY POINTER
17141  106 326 036            CALL  CWAIT      MAKE SURE THE DISPLAY IS READY
17144  006 040                LA    040        ERASE THE CHARACTER
17146  127                    EX    WRITE
17147  260                    ORA              RETURN WITH ZERO CONDITION FALSE
17150  007                    RET
                     .
                     . CRT DISPLAY ROUTINE
                     .
                     . DISPLAYS A STRING OF CHARACTERS WHICH ARE IN MEMORY STARTIN
                     . WITH THE ADDRESS GIVEN IN THE H AND L REGISTERS AND AT THE
                     . POSITION DESCRIBED BY THE D (HORZ) AND E (VERT) REGISTERS.
                     . OVERFLOW OFF THE END OF A LINE IS NOT PERMITTED.
                     . SPECIAL CONTROL CHARACTERS TERMINATE THE LINE AND ALLOW
                     . MOVEMENT OF THE CURSOR. ERASURE OF THE SCREEN OR LINE.
                     . AND ROLL-UP OF THE ENTIRE SCREEN.
                     .
                     . ENTRY VALUES:         D - HORIZONTAL CURSOR POSITION (0 TO 79
                     .                       E - VERTICAL CURSOR POSITION (0 TO 11)
                     .                       HL - FIRST CHARACTER LOCATION IN STRING
                     . EXIT VALUES:          DE - CURSOR POSITION AFTER LAST CHAR
                     .                       HL - MEMORY LOCATION AFTER TERM CHAR
                     . CONTROL CHARACTERS:   003 - END OF THE STRING
                     .                       011 - A NEW HORIZONTAL POSITION FOLLOWS
                     .                       013 - A NEW VERTICAL POSITION FOLLOWS
                     .                       015 - END OF LINE (DOES CR/LF)
                     .                       021 - ERASE TO THE END OF THE FRAME
                     .                       022 - ERASE TO THE END OF THE LINE
                     .                       023 - ROLL UP THE SCREEN ONE LINE
                     .
17151  006 341       DSPLY$* LA    0341       ADDRESS THE DISPLAY
17153  121                   EX    ADR
17154  250                   XRA              TURN OFF THE CURSOR
17155  131           DOCOM:  EX    COM1       DO THE CONTROL COMMAND
17156  106 326 036   DLOOP:  CALL  CWAIT      MAKE SURE THE DISPLAY IS READY
17161  317                   LBM              GET A CHARACTER FROM THE STRING
17162  106 353 036           CALL  INCHL      BUMP THE STRING POINTER
17165  301                   LAB              CHECK FOR CONTROL CHARACTERS
17166  044 177               ND    0177       STRIP ANY PARITY
```

```
17170  074 003                   CP     3
17172  150 265 036               JTZ    ENDOS      END OF STRING
17175  074 011                   CP     011
17177  150 300 036               JTZ    PHORZ      POSITION HORIZONTALLY
17202  074 013                   CP     013
17204  150 274 036               JTZ    PVERT      POSITION VERTICALLY
17207  074 015                   CP     015
17211  150 245 036               JTZ    ENDOL      END OF LINE
17214  074 021                   CP     021
17216  150 307 036               JTZ    EEOF       ERASE TO THE END OF THE FRAME
17221  074 022                   CP     022
17223  150 314 036               JTZ    EEOL       ERASE TO THE END OF THE LINE
17226  074 023                   CP     023
17230  150 321 036               JTZ    ROLLUP     ROLL UP THE SCREEN
17233  127                       EX     WRITE      PUT OUT THE CHARACTER
17234  303                       LAD               BUMP THE CURSOR POSITION
17235  074 117                   CP     79         UNLESS AT THE END OF THE LINE
17237  014 000                   AC     0
17241  330                       LDA
17242  104 156 036               JMP    DLOOP

17245  036 000         ENDOL:    LD     •0         RETURN CURSOR TO START OF NEXT LINE
17247  304                       LAE               BUMP THE LINE COUNTER
17250  004 001                   AD     1
17252  340                       LEA
17253  074 014                   CP     12
17255  160 265 036               JTS    ENDOS      THERE IS ROOM FOR THE NEXT LINE
17260  046 013                   LE     11         ELSE KEEP THE LINE COUNTER AT ELEVEN
17262  006 010                   LA     010        AND ROLL THE SCREEN UP ONE LINE
17264  131                       EX     COM1
17265  106 326 036     ENDOS:    CALL   CWAIT      MAKE SURE THE DISPLAY IS READY
17270  006 020                   LA     020        TURN ON THE CURSOR
17272  131                       EX     COM1
17273  007                       RET               RETURN

17274  347             PVERT:    LEM               SET THE VERTICAL POSITION
17275  104 301 036               JMP    NCHAR
17300  337             PHORZ:    LDM               SET THE HORIZONTAL POSITION
17301  106 353 036     NCHAR:    CALL   INCHL      BUMP THE STRING POINTER TO THE NXT CHAR
17304  104 156 036               JMP    DLOOP
17307  006 004         EEOF:     LA     4
17311  104 155 036               JMP    DOCOM
17314  006 002         EEOL:     LA     2
17316  104 155 036               JMP    DOCOM
17321  006 010         ROLLUP:   LA     010
17323  104 155 036               JMP    DOCOM

17326  123             CWAIT:    EX     STATUS     WAIT FOR THE DISPLAY TO BE READY
17327  101                       IN
17330  012                       SRC
17331  100 326 036               JFC    CWAIT
17334  303                       LAD               MAKE SURE CURSOR IS IN CORRECT POSITION
17335  260                       ORA               PREVENT CURSOR POSITIONS OUT OF RANGE
17336  063                       RTS
```

```
17337  Ø74 12Ø            CP     8Ø
17341  Ø23                RFS
17342  133                EX     COM2
17343  3Ø4                LAE
17344  26Ø                ORA
17345  Ø63                RTS
17346  Ø74 Ø14            CP     12
1735Ø  Ø23                RFS
17351  135                EX     COM3
17352  ØØ7                RET

17353  3Ø6       INCHL*   LAL               BUMP MEMORY POINTER UP
17354  ØØ4 ØØ1            AD     1
17356  36Ø                LLA
17357  3Ø5                LAH
1736Ø  Ø14 ØØØ            AC     Ø
17362  35Ø                LHA
17363  ØØ7                RET
17364  3Ø6       DECHL*   LAL               BUMP MEMORY POINTER DOWN
17365  Ø24 ØØ1            SU     1
17367  36Ø                LLA
1737Ø  3Ø5                LAH
17371  Ø34 ØØØ            SB     Ø
17373  35Ø                LHA
17374  ØØ7                RET
```

DONE

```
                                   .
                                   . LIBRARY CATALOG
                                   .
17404                                       SET    Ø17404
17404  Ø37                         CATW*    DC     Ø37              STARTING ADDRESS FOR LOADER
17405  Ø1Ø                                  DC     Ø1Ø
17406  34Ø                                  DC     Ø34Ø             STARTING ADDRESS COMPLEMENTED
17407  367                                  DC     Ø367
17410                              CAT*     RPT    14               SPACE FOR 14 ENTRIES
17410  Ø4Ø Ø4Ø Ø4Ø Ø4Ø                      DC     '        '
17420  Ø4Ø Ø4Ø Ø4Ø Ø4Ø                      DC     '        '
17430  Ø4Ø Ø4Ø Ø4Ø Ø4Ø                      DC     '        '
17440  Ø4Ø Ø4Ø Ø4Ø Ø4Ø                      DC     '        '
17450  Ø4Ø Ø4Ø Ø4Ø Ø4Ø                      DC     '        '
17460  Ø4Ø Ø4Ø Ø4Ø Ø4Ø                      DC     '        '
17470  Ø4Ø Ø4Ø Ø4Ø Ø4Ø                      DC     '        '
17500  Ø4Ø Ø4Ø Ø4Ø Ø4Ø                      DC     '        '
17510  Ø4Ø Ø4Ø Ø4Ø Ø4Ø                      DC     '        '
17520  Ø4Ø Ø4Ø Ø4Ø Ø4Ø                      DC     '        '
17530  Ø4Ø Ø4Ø Ø4Ø Ø4Ø                      DC     '        '
17540  Ø4Ø Ø4Ø Ø4Ø Ø4Ø                      DC     '        '
17550  Ø4Ø Ø4Ø Ø4Ø Ø4Ø                      DC     '        '
17560  Ø4Ø Ø4Ø Ø4Ø Ø4Ø                      DC     '        '
17570  Ø52                                  DC     '*'
                                   .
17571  ØØØ                         ALPFN*   DC     Ø                AUTO-LOAD PHYSICAL FILE NUMBER
                                   .
                                   . END OF PHYSICAL FILE 1
                                   .
17572  Ø4Ø Ø4Ø Ø4Ø Ø4Ø            SYMBOL*  DC                      ITEM SYMBOL STORAGE
                                   .
                                   . LOAD AND EXECUTE
                                   .
17601  1Ø6 1ØØ ØØØ                 MAUTO$*  CALL   LOAD$            LOAD THE GIVEN FILE
17604  1ØØ Ø75 ØØØ                 MAUTO:   JFC    RUN$             EXECUTE IT IF GOOD LOAD
17607  1Ø4 Ø04 ØØØ                          JMP    BOOT$            ELSE RE-LOAD THE OPERATING SYSTEM
                                   .
17612  1Ø6 112 ØØØ                 MAUT2$*  CALL   LOAD2$           LOAD DECK TWO FILE
17615  1Ø4 2Ø4 Ø37                          JMP    MAUTO
                                   .
                                   . SYMBOLIC FILE LOADER
                                   .
17620  353                         MLOAD$*  LHD                     GET PACKET ADDRESS
17621  364                                  LLE
17622  Ø46 172 Ø36 Ø37                      DE     SYMBOL           PUT THE NAME IN THE LOOKUP ITEM
17626  Ø26 ØØ6                              LC     6
17630  1Ø6 345 Ø37                          CALL   BLKTFR
17633  Ø46 Ø1Ø Ø36 Ø37                      DE     CAT              LOOK IT UP IN THE LIBRARY CATALOG
17637  1Ø6 264 Ø37                          CALL   LOOKUP
17642  Ø74 ØØ5                              CP     5                SEE IF IT IS THE THE CATALOG
17644  Ø13                                  RFZ                     ZERO FLAG FALSE IF IT ISN'T
17645  3Ø6                                  LAL                     CALCULATE THE FILE NUMBER
17646  Ø44 37Ø                              ND     Ø37Ø
17650  Ø24 Ø1Ø                              SU     CAT
```

```
17652  012                      SRC
17653  012                      SRC
17654  012                      SRC
17655  004 002                  AD      2           FIRST ENTRY IS PHYSICAL FILE TWO
17657  310                      LBA
17660  106 100 000              CALL    LOAD$
17663  007                      RET

                        .
                        . SYMBOL LOOKUP ROUTINE
                        .
17664  353              LOOKUP* LHD                 CHECK FIRST ENTRY IN TABLE
17665  364                      LLE
17666  104 334 037              JMP     LSTART
17671  066 172 056 037  LOOKPU: HL      SYMBOL      GET THE ITEM STARTING ADDRESS
17675  327              LSLOOP: LCM                 GET THE NEXT ITEM CHARACTER
17676  106 365 037              CALL    INCSWP      GET THE NEXT TABLE ADDRESS
17701  307                      LAM                 GET THE NEXT TABLE CHARACTER
17702  272                      CPC
17703  110 322 037              JFZ     LDIFF       THEY DON'T MATCH
17706  306                      LAL                 SEE IF AT THE END OF THE ENTRY
17707  044 007                  ND      7
17711  074 005                  CP      5
17713  053                      RTZ                 THE ITEM HAS BEEN FOUND IF SO
17714  106 365 037              CALL    INCSWP      GET THE NEXT ITEM ADDRESS
17717  104 275 037              JMP     LSLOOP      AND TRY THE NEXT CHARACTER
17722  306              LDIFF:  LAL                 BUMP THE TABLE POINTER TO NEXT ENTRY
17723  044 370                  ND      0370
17725  004 010                  AD      8
17727  360                      LLA
17730  305                      LAH
17731  014 000                  AC      0
17733  350                      LHA
17734  307              LSTART: LAM                 GET THE TABLE FIRST CHARACTER
17735  074 101                  CP      'A'         END OF TABLE IF IT IS NOT ALPHA
17737  063                      RTS
17740  335                      LDH                 SAVE THE TABLE ADDRESS
17741  346                      LEL
17742  104 271 037              JMP     LOOKPU      AND TRY NEXT TABLE ENTRY

                        .
                        . BLOCK TRANSFER FROM HL TO DE C CHARACTERS
                        .
17745  317              BLKTFR* LBM                 GET A SOURCE CHARACTER
17746  106 365 037              CALL    INCSWP      GET NEXT DESTINATION LOCATION
17751  371                      LMB                 PUT IT IN A DESTINATION LOCATION
17752  106 365 037              CALL    INCSWP      GET NEXT SOURCE ADDRESS
17755  302                      LAC                 DECREMENT THE COUNT
17756  024 001                  SU      1
17760  320                      LCA
17761  110 345 037              JFZ     BLKTFR      DO NEXT CHAR IF NOT ZERO
17764  007                      RET

                        .
                        . INCREMENT HL AND SWAP IT WITH DE
                        .
17765  306              INCSWP* LAL
```

```
17766 004 001          AD    1
17770 364              LLE
17771 340              LEA
17772 305              LAH
17773 014 000          AC    0
17775 353              LHD
17776 330              LDA
17777 007              RET
```

DONE