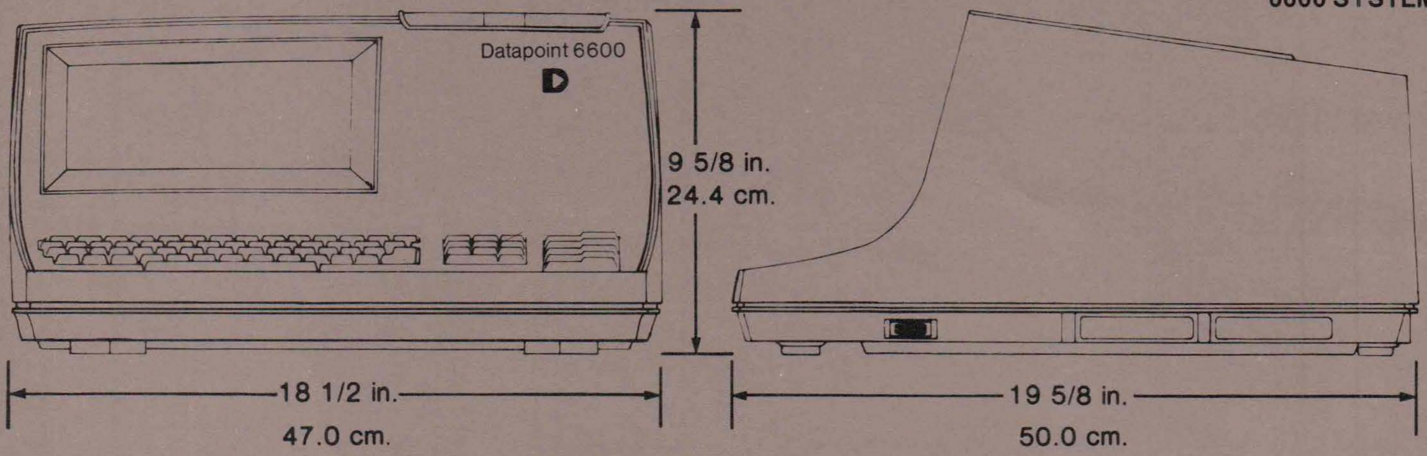
DATAPOINT CORPORATION

The leader in dispersed data processing™

Product specification and hardware reference manual

Datapoint 6600

6600 SYSTEM



June 1979
Document No. 60430

PREFACE

The computer-oriented user will find this manual useful for evaluation of Datapoint 6600 system capabilities and limitations. However, only the hardware considerations are covered in this manual. The full utility of the Datapoint 6600 system cannot be appreciated until the available software support for the machine has been reviewed.

A complete family of software packages available for the Datapoint 6600 system includes high-level languages, operating systems, source code and text editors, communications programs, utility programs, etc. Reference should be made to the latest issue of the Datapoint Software Catalog for more complete information.

TABLE OF CONTENTS

PART 1
GENERAL FEATURES

1.1 Introduction

The Datapoint 6600 is a new addition to the Datapoint family of processors. The Datapoint 6600 highlights such features as expanded memory capability to 120K user memory and faster memory and processor cycle times. The Datapoint 6600 is also completely compatible with the Datapoint 1100, 2200, 5500 and 1150.

Note: All numerics preceeded with a leading zero (0) represent an octal value.

1.2 System Elements

There are four basic elements in the 6600 system plus the capability to interface to a number of external peripheral devices.

This chapter introduces the basic elements: CRT, keyboard, processor and cassettes. Further information may be obtained from the following chapters.

1.3 CRT Display

The CRT Display provides the following features:

a. 7" x 3½" viewing area;
b. 960 characters;
c. 80-character by 12-line format;
d. Software defined 128-character font;
e. 60 frames-per-second refresh rate (50 frames-per-second when using 50 hertz power);
f. 5 x 7 matrix character generation;
g. 5 x 7 solid, blinking cursor, alternates with characters, nondestructive;
h. Single control line erasure, frame erasure, page roll-up and roll-down;
i. Direct control of all CRT functions by the processor, providing tab, editing, form control, etc; and
j. Writing rate up to 50,000 characters per second.

1.4 Keyboard

The integral keyboard provides a basic 55-key alphanumeric 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 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:

* 8-bit memory word length (plus parity)
* Complete parallel I/O system
* Automatic power-up restart

The instruction set contains all instructions used in the Datapoint 1100, 2200, 1150 and 5500 systems, providing complete upward program and input-output compatibility. In addition, the processor characteristics of the 6600 provide:

* Higher operating speed
* Hardware Multiply/Divide
* String moves
* Greater speed
* Expanded memory

This gives the 6600 considerably greater processing capability than found in the 5500 processors.

1.6 Cassette Tape Decks

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

a. 47 characters per inch density;
b. Bi-directional operation; and
c. Processor controlled data transfer, direction control, and high-speed rewind.

1.7 General Specifications

POWER REQUIREMENTS:
115 or 240 VAC (+/—10%), 60 or 50 Hz

EQUIPMENT DIMENSIONS:
Width: 18.5 in. (47 cm)
Height: 9.6 in. (24.5 cm)
Depth: 19.6 in. (50 cm)
Weight: 47 lbs. (21.3 kg)

OPERATING ENVIRONMENT: (excluding media)
10° to 38°C (50° to 100°F)
20 to 80% Relative Humidity (Non-Condensing)

1.8 Peripherals

The 6600 will accommodate a wide variety of external peripherals, such as asynchronous and synchronous communications adaptors, printers, disks, and magnetic tapes.

PART 2
KEYBOARD

2.1 General

The keyboard on the Datapoint 6600 processor performs the functions of data entry and processor control.

The integral keyboard provides a basic 55-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 processor. An audible "click" providing an acoustical feedback to the operator is available under software control.

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

The 11-key matrix may be optionally supplied with control key coding rather than numeric key coding and with keytops engraved to customer specifications.

The five control keys 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.

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.

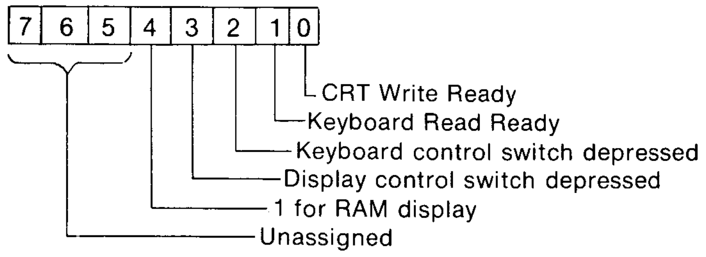
RESTART

Momentary contact switch which causes the processor to halt and executes the Restart routine contained in ROM. To protect against accidental restart, the restart function is inhibited unless the RESTART and RUN keys are depressed simultaneously.

2.2 Keyboard Operation

The keyboard is addressed by the processor by loading the A register* with 0341 octal 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 CRT/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. The keyboard is single buffered under processor control and is designed such that when a character is entered from the keyboard, another character will not be recognized from the keyboard until the processor accepts the first character entered. Bits 2 and 3 will indicate if either the KEYBOARD or DISPLAY control switch is pressed.

CRT/Keyboard Status Word



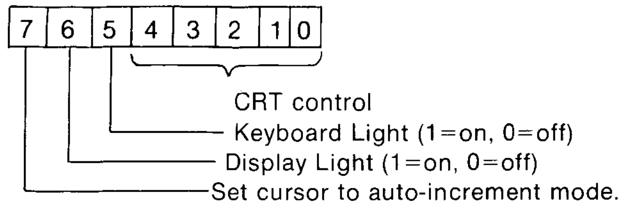
The External Commands associated with the operation of the keyboard are as follows:

a. EX BEEP. This command produces a 1500 Hertz tone for a duration of about 400 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:

CRT/Keyboard Control Word



7 6 5 4 3 2 1 0
CRT control
Keyboard Light (1=on, 0=off)
Display Light (1=on, 0=off)
Set cursor to auto-increment mode.

Note: The CRT Write Ready must be true before the EX COM1 can be issued.

* For I/O transfers in the 6600, the A register is used if another register is not specified. See Part 5, category 2, for further information.

TABLE 2-1
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	^-136
K-113	k - 153	Space-040	___-137
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	Del - 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	

SPECIAL NUMBER PAD OPTION

(.)-016
(0)-020
(1)-021
(2)-022
(3)-023
(4)-024
(5)-025
(6)-026
(7)-027
(8)-030
(9)-031

*These codes are all represented in octal

PART 3
DISPLAY

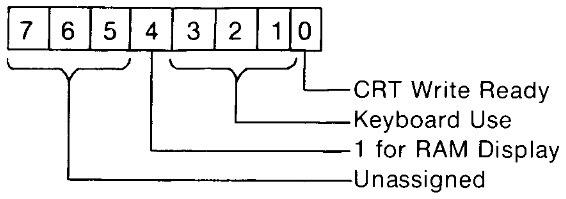
3.1 General Description

The 6600 display provides extended character generation flexibility and fast character transfer rates. The display system includes: CRT Display of 12 lines of 80 characters, power line screen refresh rate, 960 cells of random access memory holding the screen image, a program loadable random access character generation memory capable of producing 128 individual 5 by 7 dot matrix characters, a group of registers utilized to position the cursor, and automatic cursor increment provisions. The maximum character transfer rate to the CRT is determined by processor input/output speed. The upper limit of the display transfer rate is approximately 50,000 characters per second.

3.2 Display Operation

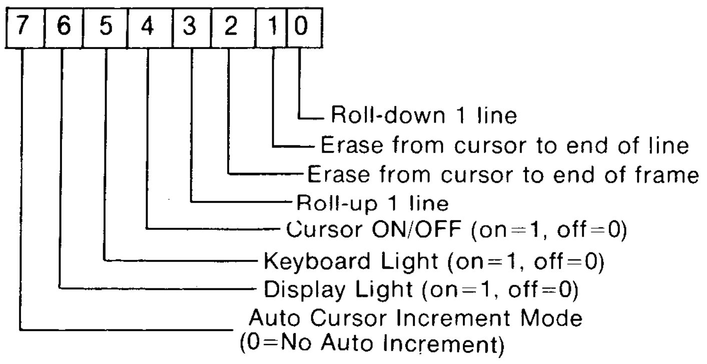
The CRT is addressed by the processor by loading the A register with octal 0341 and executing an EX ADR command. (Note that the keyboard also uses this address, see Part 2.) Following the address sequence, the CRT/keyboard status word can be loaded into the A register by executing an INPUT instruction. The CRT status assignment is as follows: Bit 0 of the status word indicates that the CRT is ready to accept data or commands if it is set to a logical 1. (Note that this status bit will indicate a logical one if the cursor is positioned to an invalid screen position.) Bits 1, 2 and 3 are used for keyboard status.

CRT/Keyboard Status Word



Control of the CRT is accomplished through the use of the following external commands:

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:


7 6 5 4 3 2 1 0
Roll-down 1 line
Erase from cursor to end of line
Erase from cursor to end of frame
Roll-up 1 line
Cursor ON/OFF (on=1, off=0)
Keyboard Light (on=1, off=0)
Display Light (on=1, off=0)
Auto Cursor Increment Mode
(0=No Auto Increment)

The following explanations assume that the CRT has been addressed.

BIT 0: Each execution of EX COM1 with this bit set to 1 causes the roll-down operation to occur. All displayed characters (not the cursor) are moved down one line. The bottom line on the screen is lost and the top line is filled with the pattern in position 040 octal of the character generation memory. The Write Ready status bit goes false until the roll-down operation is complete; another EX COM1 must not be issued during this time.

BIT 1: Each execution of EX COM1 with this bit set to 1 causes erasure from (including) the current cursor position to the end of the line. The character displayed in the erased positions is determined by the pattern in position 040 octal of the character generation memory. The Write Ready status bit goes false until this operation is complete; another EX COM1 must not be issued during this time.

BIT 2: Each execution of EX COM1 with this bit set to 1 causes erasure from (including) the current cursor position to the end of the frame. The character displayed in the erased position is determined by the pattern in position 040 octal of the character generation memory. The Write Ready status bit goes false until this operation is complete; another EX COM1 must not be issued during this time.

BIT 3: Each execution of EX COM1 with this bit set to 1 causes the roll-up operation to occur. All displayed characters (not the cursor) are moved up one line. The top line on the screen is lost and the bottom line is filled with the pattern in position 040 octal of the character generation memory. The Write Ready status bit goes false until the roll-up operation is complete; another EX COM1 must not be issued during this time.

BIT 4: 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, and will be turned on again when RUN is depressed if the cursor was on prior to the HALT.

BITS
5, 6: Keyboard & Display Light — See Part 2.

BIT 7: When this bit is set to 1, the automatic cursor increment feature is in effect. In auto cursor increment mode, the cursor moves one character to the right after each EX WRITE command. The vertical position of the cursor does not change. If the last character (horizontal position 79) is written, the cursor will increment off the screen and the CRT Write Ready status bit will stay true until the cursor is re-positioned back onto the screen.

b. EX COM2 (Command 2) positions the cursor to the horizontal character slot designated by the contents of the A register. Character positions 0-79 (decimal) or 0-0117 (octal) are valid.

c. EX COM3 (Command 3) positions the cursor to the line designated by the contents of the A register. Line numbers 0-11 (decimal) or 0-013 (octal) are valid.

d. EX COM4 (Command 4) places the character generator memory in the load mode and sets the load pointer to the contents of the A register. Character positions 0-127 (decimal) or 0-0177 (octal) are valid.

e. EX WRITE transfers the character in the A register to the screen image memory at the position indicated by the cursor position. The cursor need not be on for this transfer to occur. If the auto cursor increment feature is enabled the cursor position will be incremented after the transfer. When the character generation memory has been set to the load mode, the above transfer is inhibited (as is the automatic cursor increment) and EX WRITE transfers data from the A register to the character generation memory. Execution of an EX WRITE (to either the screen image memory or the character generation memory) causes the Write Ready status bit to go false for up to 17 microseconds. Unless a delay of at least this duration is guaranteed by the program, the Write Ready status bit should be checked before execution of an EX WRITE, EX COM1, EX COM2, EX COM3 or EX COM4 after a previous EX WRITE. Note that EX COM2 and EX COM3 do not affect the Write Ready status.

Five successive byte transfers are required to load a complete 5 by 7 character dot pattern. The loading format is illustrated by the following diagram which illustrates the letter "A" loaded into memory:

Bit No.					
6		x	x	x	
5	x				x
4	x				x
3	x	x	x	x	x
2	x				x
1	x				x
0	x				x
	1	2	3	4	5

Transfer Number (EX WRITE)

For example, the procedure for loading the character location 0101 with an "A" as illustrated would consist of the following character transfers:

```
•
•
•
LA      0101      Set load pointer to
EX      COM4      Location 0101
LB      077
CALL    DWRITE    Load column 1
LB      0110
CALL    DWRITE    Load column 2
CALL    DWRITE    Load column 3
CALL    DWRITE    Load column 4
LB      077
CALL    DWRITE    Load column 5
•
•
•
```

The DWRITE subroutine below is used here instead of an EX WRITE instruction to guarantee the 17 microseconds delay required between executions of EX WRITE instructions:

```
            •
            •
DWRITE      LAB
            EX        WRITE
DWRITW      IN
            SRC
            JFC       DWRITW
            RET
```

After all five columns of a character have been loaded, the character load pointer is automatically incremented to the following character. In the case of the above example the load pointer will be incremented to location 0102. Note that it is only necessary to issue additional EX COM4, when nonsequential character locations are being loaded. The display logic card is removed from the load mode by the execution of an EX COM1 (with A=0 if no other function is desired).

As mentioned previously, the Write Ready status bit goes false during the roll-up, roll-down, erase-to-end-of-line and erase-to-end-of-frame operations. The maximum periods during which Write Ready will be false for each of these operations is tabulated below for 60 Hz and 50 Hz primary power frequency:

OPERATION	50HZ	60HZ
Roll up	21.1 msec	17.8 msec
Roll down	21.1 msec	17.8 msec
Erase to-end-of-line	21.1 msec	17.8 msec
Erase-to-end-of-frame	35 msec	31.7 msec

PART 4

CASSETTE TAPES

4.1 General Description

The Datapoint 6600 contains two cassette tape recording devices for storage of programs and data. Since the hardware Restart (Appendix A, 1.3) uses the rear deck (number one), programs will typically be on it while data areas will be on 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. See Table 4-1 for a list of physical specifications.

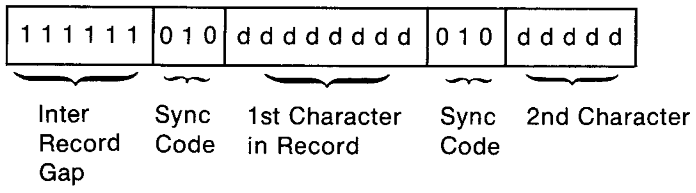
4.2 Operations

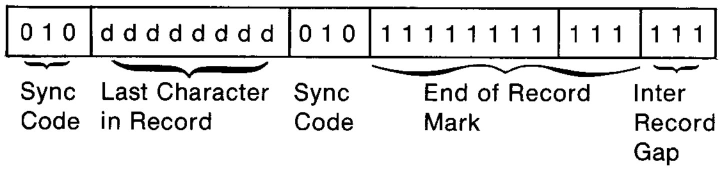
Data is recorded or read in bit serial fashion on one track. Each eight bit character is framed by three sync bits on either side of the character.

The first eight bit string framed by valid sync code groups (010) indicates the beginning of a record. The appearance of eleven ones in a row indicates the end of a record. Sync code groups after the first character in a record and before the end of the record are ignored.

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 appear reversed (bit 0 will be bit 7, 1 will be 6, etc.)

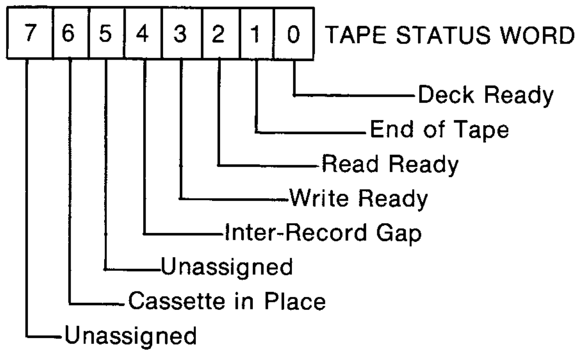
This is what a typical record looks like:





4.3 status

The cassette tape unit is addressed by the processor by loading the A register with 0360 octal 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:



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). When Deck Ready is true the tape will be stopped, a cassette in the selected deck, and the head engaged. 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 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.

4.4 Control (Table 4-2)

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 and 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 DECK 1 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 DECK 2 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 of a valid character. When an invalid sync code is encountered 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 msec. 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 EX TSTOP will stop the motion initiated by EX SF.

g. EX SB is similar to EX SF except that tape motion is in the reverse direction and the data bits are reversed.

h. EX WBK causes the currently selected deck to be set in forward motion and all status bits except the Write Ready to go false. A character must then be presented within 2.8 msec. (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 signaled to the hardware by withholding a character for a period of time longer than the 2.8 msec. 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.

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

j. PUNCH TABS on the cassette cartridge are used for "write protect" and "automatic restart." The punch tab on the left (as you face the processor) 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 4-1

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-Record Gap)	2 inches
Rewind Speed	90 ips
Rewind Time (max 300 ft.)	40 sec.
Character Transfer Time	2.8 msec.

TABLE 4-2

COMMAND NUMBER		OCTAL CODE	COMMAND	DESCRIPTION	DEVICE ADDRESS
15	DECK1	155	Select Deck 1	Connects deck 1 to I/O bus	0360
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	0360
19	——	165	(Unassigned)	——	——
20	BSP	167	Backspace One Block	Backs up the selected tape one record	
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	0360

PART 5
PROCESSOR

The processor in the 6600 is comprised of two sets of eight 8-bit program accessible registers, two sets of 4 control flags, 128K bytes of memory (120K bytes of user program memory), a 16-bit program counter, an 8-bit instruction register, an 8-bit base register, a 16-level push down Stack, a special 4-bit instruction modification register and a 16-word memory sector table.

5.1 Processor Registers

The eight programmable registers are named A, B, C, D, E, H, L, and X. The flag flip-flops are named C (carry), Z (zero), S (sign), and P (parity). There are two sets of these registers and flags and access to them depends upon the mode the processor is in. Upon Restart or whenever the Alpha mode instruction is executed, all Alpha mode registers and flags are accessible by the program. Whenever a Beta mode instruction is executed, the Beta mode registers and flags are accessible. No other registers or functions within the machine are affected by the processor mode.

Registers A-L are general purpose registers which may be interchanged with each other as to their functions. When an arithmetic, logical or I/O instruction is performed and a register is not specified, the "A" register is over stored with the result.

When using registers for addressing, they may be paired together to form a 16-bit address; XA, BC, DE and HL. If a pair of registers is not specified, the HL registers will be assumed.

The X register is a working page register and is not normally used for the same functions as registers A-L, except to form the upper 8 bits of a 16-bit address word.

P - The P register is the "location counter" for the program and 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 16 bits wide.

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. I is 8 bits wide. This register is for internal hardware sequencing and is transparent to the user.

5.2 Comparison With Datapoint 5500 and 2200 Systems

5.2.1 Input/Output

Besides simply executing I/O instructions faster than the 5500 and 2200 systems, the 6600 system I/O has parity checking while maintaining control over compatibility with 5500 and 2200/1100 devices.

5.2.2 Input Parity Checking

A ninth wire exists in the input and output data paths of the I/O bus. An INPUT instruction (PIN) exists which will cause an interrupt if there is not an odd number of ones out of the nine bits on the input bus when the data is strobed into the processor. Note that if a non-existent device is addressed and then a PIN is executed, a parity fault will occur because the status will be nine zeros, which is an even number (zero) of ones. Also note that using the INPUT instruction will never cause a parity fault interrupt, allowing all 2200 programs to execute properly on the 6600 systems (see Section 5.2.4).

5.2.3 Output Parity Checking

In addition to the output bus parity bit, there is another input wire to the processor called the Output Parity Fault line. If this wire is low during the parity fault check window (about 40 nanoseconds wide occurring 2 to 6 microseconds after the trailing edge of any output strobe), the output parity fault interrupt will occur. A 6600 system I/O device can check for an even number of ones out of the nine output bits at the leading edge of the output strobe. If there are an even number of ones, the device can hold the Output Parity Fault line low until the leading edge of the next I/O strobe, thus causing the Output Parity Fault interrupt.

5.2.4 Compatibility With 5500 and 2200 Systems Peripherals

6600 system peripherals may not be directly compatible with the 2200 because of the use of output parity checking, but are directly compatible with the 5500. However, 2200 peripherals can be made to work on the 6600 system if the PIN (parity checking input) instruction is not used. Also 6600 system peripherals may be used on 2200 systems via an I/O option strap. The three additional wires used in the 6600 system I/O bus are not used in the 2200 system I/O.

5.3 Memory

In addition to having more memory capability than the 5500 system, the 6600 memory system is faster. The 6600 also features parity checking and advanced memory handling.

5.3.1 Parity Checking

Each byte in the memory system has a ninth bit which is used for parity checking. Even parity is written into every location automatically when the machine is powered up and into the given location whenever a data byte is written (the words are written such that there are always an even number of ones out of the total number of nine bits). Whenever a data byte is read, a check for even parity is made and a special interrupt invoked if the check fails. This interrupt supplies the logical address of the failing memory location for diagnostic purposes. This means that the base addressing of the particular routine being run would have to be known to convert the failure address to a physical memory address. Note that if a non-existent memory location is accessed, a parity fault will not occur because all zeros (even number of ones) will be read. In addition to the RAM, the 6600 contains a ROM (read-only memory) which is used for power initialization, RESTART, debugging, memory testing, and other system functions. The parity bit for ROM is generated artificially.

5.3.2 Physical Layout

The 6600 contains provisions for five memory boards. The first four boards contain 32K bytes of RAM each. The fifth board contains 4K of ROM and overlays locations 0170000 through 0177777. This gives 124K of RAM, 4K of which is reserved for System RAM, leaving a total of 120K of user RAM.

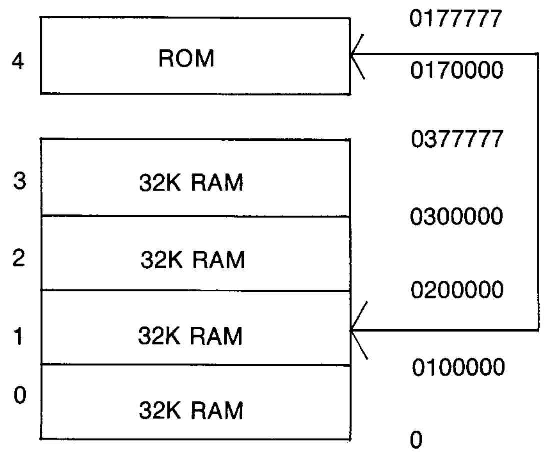


Figure 5-1
MEMORY LAYOUT

5.3.3 Address Generation

Figure 5-1 is a map of the physical memory layout. This memory is referenced by what is called a "physical" memory address. Board 1 is physical locations 0 through 077777 (RAM), board 2 is physical locations 0100000 through 0167777 (RAM), board 3 is physical locations 0200000 through 0277777 (RAM), board 4 is physical locations 0300000 through 0377777 (RAM), and board 5 is physical locations 0170000 through 0177777 (ROM).

User programs use what is called a "logical" memory address. This is a 16-bit value created by the program and translated to the proper "physical" memory address by a mechanism in the processor. The translation mechanism utilizes a base register and a memory sector table as depicted in Figure 5-2.

If the logical memory address is between 0100000 and 0137777, its upper eight bits are added (two's complement) to the eight bit base register. Otherwise, the upper eight bits of the logical memory address are unchanged by the adder. The new 16-bit value consisting of the lower eight bits of the logical memory address and the eight bits from the adder is called the "based logical memory address." Note that the base register may be negative (two's complement) for creating based logical memory addresses lower than 0100000.

The upper four bits of the based logical memory address form an address for the 16-entry 8-bit sector table. This table divides the 64K based logical memory space into sixteen 4K byte sectors, each of which may be translated to any physical 4K memory section and may be protected from being accessed if the USER mode flag is set or from being written into regardless of the state of the USER mode flag. (Note that many people in the computer industry refer to the sector table as a page table. However, the reference has been changed here to avoid confusion with the term "page" used elsewhere to denote a 256 byte section of logical memory space starting at an address of 0 modulo 256.)

The sector table contains eight bits for each entry. Bit 1 (the next to the least significant) of a sector table entry contains a hardware generated and checked parity bit. Any value loaded into this position is ignored since the hardware generates the proper parity bit when a sector table entry is loaded. If, during any memory access, there are not an odd number of one bits out of the eight sector table entry bit positions, a Sector Table Parity Error System Call interrupt will be generated to memory location 0167474. Bit 2 of a sector table entry is set to enable the sector to be accessed (read or written) when the machine is in User Mode. Bit 3 of a sector table entry is set to enable the sector to be written in either User or System Mode. Bits 4 through 7 of a sector table entry are used for physical memory address bits 12 through 15 and bit 0 of a sector table entry is used for physical memory address bit 16 (giving a total of 17 bits of physical memory address to allow accessing 128K of physical memory space.)

With the address generation mechanism described above, two major benefits can be realized. The first is ease of reentrant coding for multiple user tasks. The program can load into the base register the base address (in multiples of 256 bytes) of his non-reentrant data area minus 0100000 and then all references to logical memory addresses between 0100000 and 0100000 plus the length of his data area will automatically be translated into the proper based logical memory location. The second major benefit is afforded by the sector table. Besides providing the ability to implement a completely protected monitor, the sector table provides ease in running several independent partitions in memory at once.

5.4 Pushdown Stack

A feature of the 6600 is the incorporation into the processor's structure of a pushdown Stack. This is useful for subroutine calling, saving the value of register pairs, calculating an address and then jumping to it without having to overstore a JUMP instruction, making an abortive exit from a subroutine (returning control to a location other than the one after the CALL instruction), and saving the state of the machine (if there is at least one free stack location).

Information may be transferred between either the P-counter and the Stack or any register pair and the Stack. The Stack is actually a separate scratch pad memory of sixteen 16-bit words which is addressed by a four-bit up/down counter. Whenever a CALL or PUSH instruction is executed, the P-counter or indicated register pair is written into the Stack word pointed out by the Stack Pointer which is then incremented. The pointer ends-around to 0 if it is incremented past 15. Whenever a RETURN or POP instruction is executed, the Stack pointer is first decremented (ending around to 15 if it is decremented below 0) and then the P-counter or indicated register pair is loaded from the pointed location. Note that the above description implies that the maximum subroutine nesting depth is sixteen and will be less if data is also pushed onto the Stack. That is, the seventeenth CALL or PUSH will overstore the value written in the first if no RETURN or POP instructions intervene.

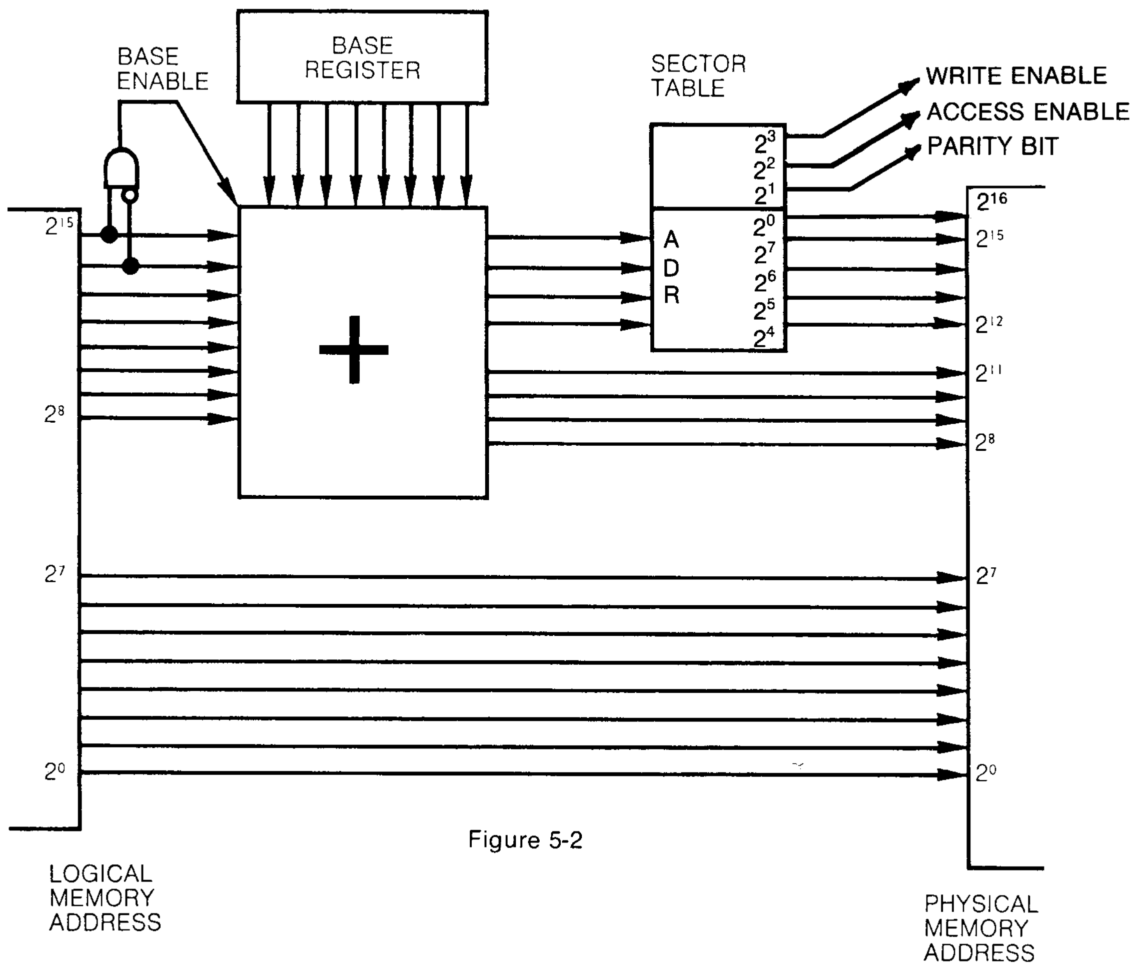
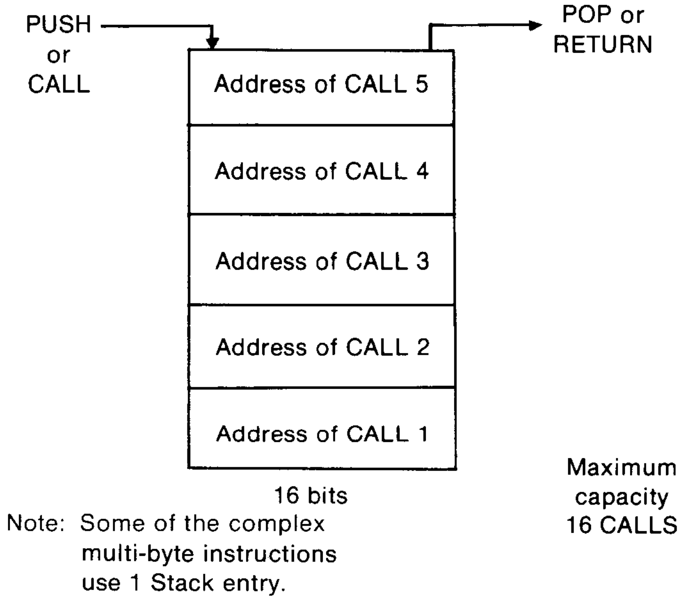


Figure 5-2



5.5 Control Flip-Flops

Also contained in the basic processor are eight control (flag) flip-flops (four in ALPHA mode and four in BETA mode) which reflect the state of the arithmetic logic unit and which can 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-Carry flip-flop. Set when an arithmetic operation results in either a carry (add) or borrow (subtract).

Z - Zero flip-flop. Set when the result of an arithmetic or logical operation is equal to zero.

S - Sign flip-flop. Reflects the state of bit 7 after an arithmetic or logical operation.

P - Parity flip-flop. Indicates parity after any arithmetic or logical operation. This is entirely separate from the I/O or memory parity system referred to elsewhere. If this flip-flop is set (true) there are an odd number of one bits; if it is reset (false), there are an even number of one bits.

5.6 System ROM Functions

See Appendix A for a complete description of the 6600 processor ROM features.

5.7 Interrupt Handling

There are eleven different interrupt events possible in the 6600. All except the power-up interrupt use the System Call mechanism (see instruction description) to the memory location explained below. The System Call mechanism pushes the current value of the P-counter onto the Stack, disables the one millisecond interrupt, clears the USER mode and forces execution to continue at the indicated vector location. Note that one of the interrupts is actually the SYSTEM CALL (SC) instruction and that the other interrupts use the same mechanism but jump to different locations.

The following describe the interrupt vector entry point locations. Note that all of these vectors are in System RAM locations and are initialized on power-up. See Appendix A for a description of how those are handled in the system ROM.

0167400 MEMORY PARITY FAULT

This is caused by a memory read resulting in a nine bit word with an odd number of ones. Before the P-counter was pushed onto the Stack by the System Call mechanism, the *based logical* memory address of the faulty memory cell was pushed onto the Stack.

Note that during multiple byte operations which use the Stack, the P-counter is used during the instruction to hold a data address. If an interrupt occurs during one of these instructions, the value the P-counter pushed onto the Stack will be a data address instead of the true P-counter value (the actual P-counter value being another entry further down on the Stack). For this reason, one cannot always determine the state of the machine if an interrupt occurs.

0167406 INPUT PARITY FAULT

This is caused by a PIN or MIN instruction (see instruction explanation) resulting in a nine bit word from the I/O Bus with an even number of ones. The P-counter value pushed onto the Stack points to the PIN or MIN instruction.

0167414 OUTPUT PARITY FAULT

This is caused by the Output Parity Fault line on the I/O Bus being low during the parity fault check window (about 40 nanoseconds occurring 2 to 6 us after the trailing edge of any output strobe). The Output Parity Fault line can be held low by 6600 System I/O devices if they see an even number of ones out of the nine bits of the I/O Bus. The P-counter value pushed onto the Stack points to the output or MOUT instruction.

0167422 WRITE PROTECT VIOLATION

This is caused by a memory write operation being attempted on a sector of memory for which the Write Enable bit (A3 in the sector table entry) has not been set.

Note that during multiple byte operations which use the Stack, the P-counter is used during the instruction to hold a data address. If an interrupt occurs during one of these instructions, the value the P-counter pushed onto the Stack will be a data address instead of the true P-counter value (the actual P-counter value being another entry further down on the Stack). For this reason, one cannot always determine the state of the machine if an interrupt occurs.

0167430 ACCESS PROTECT VIOLATION

This is caused by the USER mode flag being set and a memory operation being performed on a sector of memory for which the access enable bit (A2 in the page sector table entry) has not been set. The same note concerning multiple byte operations and the Memory Parity Fault interrupt applies to the access protect violation interrupt.

0167436 PRIVILEGED INSTRUCTION VIOLATION

This is caused by the execution of an I/O instruction or an instruction capable of changing the sector table or base register while the USER mode flag is set. The P-counter value pushed onto the Stack points to the instruction which caused the interrupt.

0167444 ONE MILLISECOND INTERRUPT

This is caused every 1000 microseconds. These interrupts can be inhibited with the DI instruction as in the 5500 system (and are inhibited with RESTART or POWERUP).

0167452 USER SYSTEM CALL

This is caused by the execution of an SC instruction.

0167460 BREAK POINT

This is caused by the execution of a BP instruction.

0167466 UNDEFINED INSTRUCTION

This is caused by an attempt to execute an instruction which is undefined in the 6600.

0167474 SECTOR TABLE PARITY FAULT

This is caused when a parity error is detected while loading in the Sector Table during any memory access.

5.8 Processor Instructions

The 6600 processor instructions have been divided into seven categories for convenience of presentation.

* Category one: All instructions contained in 1100 and 2200 system processors.
* Category two: 2200 system instructions which have been enhanced with additional register referencing capability.
* Category three: Multi-byte (string) instructions.
* Category four: Instructions for saving and restoring the state of the processor.
* Category five: Address manipulation instructions.
* Category six: Operating system control instructions.
* Category seven: 6600 Instruction
* Set and Instruction Timing.

5.8.1 Comparison to 2200 System Instructions

The 6600 has a number of instructions not in 2200 system processors. Before these instructions can be described, however, the new data paths in the processor must be described. A new discrete register (not part of the register stack containing the general purpose registers) has been added. It is a working register called the *implicit register.*

Many 2200 instructions reference the A register implicitly (e.g., use it for an accumulator or load it from the I/O Bus). The register that is implicitly referenced in the 6600 in these cases is still the A register unless an instruction is executed which changes the implicitly referenced register for the following instruction *only*. There are eight instructions (one byte long) which allow the implicit register to be loaded with one through eight (implying registers A, B, C, D, E, H, L, or X). Once this is done, interrupts are inhibited until the following instruction is completed. If the following instruction would reference the A register implicitly in the 2200, the 6600 will reference the register indicated by the implicit register value instead. This also applies to instructions where HL is the implied register pair specifying an address. The implicit register can be used to specify a different register pair (implying register pairs BC, DE, HL or XA). Notice the use of the word "implied", as references are made to the "implied register" in later descriptions.

The instructions which set the implicit register will not be described separately since they are used only to augment the function code (op code) of the instruction which they modify. In some cases the value of the implicit register will not determine a register reference but will modify an operation action instead. The implicit register is also used for a loop counter in many of the multi-byte instructions. Since the implicit register is only 4 bits wide the multibyte instructions that use it for a loop counter are limited to executing the loop sixteen times (usually meaning that fields are limited to sixteen bytes in width). However, some of the multi-byte instructions use a general purpose register for a loop counter enabling them to loop 256 times. The one millisecond interrupt can occur only during the fetch of a new instruction if interrupts are enabled at all. This means that for some of the longer multi-byte instructions, interrupts can be disabled for as long as 840 microseconds. This would be troublesome if one was using the one millisecond clock for short-term time critical work. The full 256 byte capability is included, however, in the event that one might find it useful if time critical work was not being performed.

Two additional general purpose registers have been added to the 6600 processors. By general purpose, it is meant that there is one for each mode (ALPHA and BETA) and that they reside in the register stack along with the rest of the general purpose registers. In the 6600 this register (numbered 7 in the general purpose register stack) is called the X register.

The X register is not quite as generally accessible as the rest of the registers, due to the fact that register select number 7 is used to specify memory in many instructions. However, the X register can be loaded immediately as well as be accessed via the implicit register mechanism and also by several instructions which use the X register's contents as the upper eight bits of an address. The X register is generally used in the 6600 system to indicate a working page in memory. (Here, the word "page is used to denote a 256 byte section of *logical* memory space.)

The use of the X register enables several of the instructions which provide a fixed memory address in the instruction to be one byte shorter by not having to specify the upper eight bits of the address (using the contents of the X register instead). Experience in programming the 2200 system has shown that one working storage page is generally quite adequate to hold most of the items accessed most often by a given program and that these items are accessed often enough to make the X register concept useful both in terms of saving memory and increasing speed.

Additional programming conventions developed with the 2200 system have been reflected in the 6600 instruction set. The BC and DE register are often used as pairs to form a sixteen bit value (B or D being the MSP and C or E being the LSP). Several of the new instructions treat these pairs specifically as sixteen bit values.

5.8.2 Presentation Format

A description of each 6600 instruction is given below. In order to simplify the presentation, the following symbols and abbreviations are used:

Operation:	Symbolic representation of instruction description
Op Code:	Operation Code, expressed in octal
Timing:	Execution time in microseconds (Note: memory refresh overhead is 5% implying that a program will execute, on the average, 5% slower than the sum of the indicated timings.)
Length:	Number of bytes in the instruction (Used when the length may not be especially obvious from the op code or the instruction diagram.)
Stack:	Number of stack entries
Entry:	Conditions necessary before execution
Exit:	Conditions existing after execution
Algorithm:	Steps taken to perform the instruction execution
( )	The contents of
$\leftarrow$	Is replaced by
$\rightarrow$	Is transferred to
:	Is compared with
V	Logical "Or" operation
$\veebar$	Logical "Exclusive Or" operation
$\wedge$	Logical "AND" operation
A B C D E H L X	8-bit processor registers
M	Contents of Memory location designated by the contents of HL or the designated register pair
P	Program counter (When shown P + X location relative to first byte of instruction)
Stack	The Pushdown Stack
(OP)	One of the eight ALU operations (AD, AC, SU, SB ND, XR, OR, CP)
(rs)	A source general register (ABCDEHL)(s=0 to 6)
(rd)	A destination general register (ABCDEHL) (d=0 to 6)
(r)	A general register (ABCDEHLX) (s or d =0 to 7)
(rp)	One of the pairs of registers (BC DE HL XA)
r	A register select op code No byte is necessary for selection of the A register Otherwise: B=0111, C=062, D=0113, E=0174, H=0115 L=0176, X=022
rp	A register pair select op code No byte is necessary for the selection of HL Otherwise: BC=062, DE=0174, XA=022
rp+1	BC=0113, DE=0115, HL=0117, XA=0111
(vvv)	An 8-bit value used in an instruction
(adr)	A 16-bit value used in an instruction with the LSP first, followed by the MSP
(cf)	Control flags (CZSP) (c=0 to 3) (Often called flip-flops)
(exp)	External command, listed in Table 5-1
data	An expression reducing to an 8-bit immediate value
loc	An expression reducing to a 16-bit address

5.8.3 Category 1 — 2200 System Instructions

For timing, refer to 5.8.10

LOAD IMMEDIATE **L (r)**

Op Code: 0d6 (vvv)

Operation: (vvv)→(r)

Transfers the value of the operand given in the instruction to the register specified by bits 3-5 of the instruction word.

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

1. d is the destination designator.
2. None of the flag flip-flops are changed.

LOAD **L(rd)M, L(rd)(rs), LM(rs)**

For L(rd)M: Op Code: 3d7

Operation: (M)→(rd) d≤6

For L(rd)(rs): Op Code: 3ds

Operation: (rs)→(rd) s≤6, d≤6

For LM(rs): Op Code: 37s

Operation: (rs)→(M) s≤6

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.

7 6	5 4 3	2 1 0
3	d	s

1. The source data is unaffected.
2. s and d both = 7 results in a HALT instruction.
3. None of the flag flip-flops are changed.

ADD IMMEDIATE **AD data**

Op Code: 004 (vvv)

Operation: (A) + (P+1)→A

Adds the value of the (data) operand to the contents of the A register and retains the sum in the A register.

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

1. Carry flip-flop set if add overflow occurs; otherwise carry is reset.
2. The Sign, Zero and Parity flip-flops indicate the status of the A register at completion.

ADD **AD(rs), ADM**

For AD(rs): Op Code: 20s

Operation: (A) + (rs)→A

For ADM: Op Code: 207

Operation: (A) + (M)→A

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

7 6	5 4 3	2 1 0
2	0	s

s specifies the operand source.

ADD WITH CARRY IMMEDIATE **AC data**

Op Code: 014 (vvv)

Operation: (A) + (P+1) + (Carry)→A

Adds the Carry bit and contents of the operand to the contents of the A register and retains the sum in the A register.

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

1. If add overflow occurs, the Carry flip-flop is set; otherwise Carry is reset.
2. The Sign, Zero and Parity flip-flops indicate the status of the A register at completion.

ADD WITH CARRY **AC(rs), ACM**

For AC(rs): Op Code: 21s

Operation: (A) + (Carry) + (rs)→A

For ACM: Op Code: 217

Operation: (A) + (Carry) + (M)→A

This instruction is identical to ADD WITH CARRY IMMEDIATE with the exception of operand source.

7 6	5 4 3	2 1 0
2	1	s

s specifies the operand source.

SUBTRACT IMMEDIATE **SU data**

Op Code: 024 (vvv)

Operation: (A) - (P+1)→A

Subtracts the value of the operand from the contents in the A register and retains the difference in the A register.

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

1. The Carry flip-flop is set if underflow occurs, otherwise carry is reset.
2. The Zero, Sign and Parity flip-flops represent the status of the A register at completion.

SUBTRACT **SU(rs), SUM**

For SU(rs): Op Code: 22s

Operation: (A)-(rs)→A

For SUM: Op Code: 227

Operation: (A)-(M)→A

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

7 6	5 4 3	2 1 0
2	2	s

s specifies the operand source.

SUBTRACT WITH BORROW IMMEDIATE **SB data**

Op Code: 034 (vvv)

Operation: (A)-(P+1) - (Carry)→A

Subtracts the value of the operand and the Carry bit from the contents of the A register, and retains the difference in the A register.

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

1. Sets the Carry flip-flop if underflow occurs; otherwise resets Carry.
2. The Zero, Sign, and Parity flip-flops represent the status of the A register at completion.

SUBTRACT WITH BORROW **SB(rs), SBM**

For SB(rs): Op Code: 23s

Operation: (A)-(rs)-(Carry)→A

For SBM: Op Code: 237

Operation: (A)-(M) - (Carry)→A

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

7 6	5 4 3	2 1 0
2	3	s

s specifies the operand source.

AND IMMEDIATE **ND data**

Op Code: 044 (vvv)

Operation: (A)$\wedge$(P+1)$\rightarrow$A

Forms the logical product of the contents of the A register with the value of the operand and places the result in the A register.

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

1. Resets the Carry flip-flop upon completion.
2. The Zero, Sign and Parity flip flops represent the status of the A register upon completion.

Sample Operation:

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

AND **ND(rs), NDM**

For ND(rs): Op Code: 24s

Operation: (A)$\wedge$(rs)$\rightarrow$A

For NDM: Op Code: 247

Operation: (A)$\wedge$(M)$\rightarrow$A

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

7 6	5 4 3	2 1 0
2	4	s

s specifies the operand source.

OR IMMEDIATE **OR data**

Op Code: 064 (vvv)

Operation: (A) V (P+1)$\rightarrow$ A

Forms the logical sum of the contents of the A Register and the value of the operand, and places the result in the A register.

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

1. Resets the Carry flip-flop upon completion.
2. The Zero, Sign and Parity flip-flops represent the status of the A register upon completion.

Sample Operation:

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

OR **OR(rs),ORM**

For OR(rs): Op Code: 26s

Operation: (A) V (rs)$\rightarrow$A

For ORM: Op Code: 267

Operation: (A) V (M)$\rightarrow$A

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

7 6	5 4 3	2 1 0
2	6	s

s specifies operand source.

EXCLUSIVE OR IMMEDIATE **XR data**

Op Code: 054 (vvv)

Operation: (A)$\veebar$(P+1)$\rightarrow$A

Forms the logical difference of the contents of the A register and the value of the operand, and places the result in the A register.

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

1. Resets the Carry flip-flop at completion.
2. The Zero, Sign and Parity flip-flops represent the status of the A register upon completion.

Sample operation:

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

EXCLUSIVE OR **XR(rs), XRM**

For XR(rs): Op Code: 25s

Operation: (A) $\veebar$ (rs)$\rightarrow$A

For XRM: Op Code: 257

Operation: (A) $\veebar$ (M)$\rightarrow$A

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

7 6	5 4 3	2 1 0
2	5	s

s specifies the operand source.

COMPARE IMMEDIATE **CP data**

Op Code: 074 (vvv)
Operation: (A) : (P+1)

Compares the contents of the A register with the value of the operand.

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

1. The flag flip-flops assume the same state as they would for a Subtract instruction.
2. The contents of the A register are unaffected.

COMPARE **CP(rs), CPM**

For CP(rs): Op Code: 27s
Operation: (A):(rs)
For CPM: Op Code: 277
Operation: (A):(M)

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

7 6	5 4 3	2 1 0
2	7	s

s specifies the operand sources

UNCONDITIONAL JUMP **JMP loc**

Op Code: 104 (adr)
Operation: (adr) ⟶ P

An unconditional transfer of control. The second byte of the instruction represents the least significant portion of the jump address, while the third byte of the instruction represents the most significant portion.

			P+1	P+2
7 6	5 4 3	2 1 0	7 0	7 0
1	0	4	LSP	MSP
Op Code			Address	

JUMP IF CONDITION TRUE **JT(cf) loc**

Op Code: 1(c+4) 0 (adr)
Operation: If condition true, (adr) ⟶ P

Examines the designated flip-flop. If set, transfers control to (adr). If reset, executes the next sequentially available instruction.

			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	

1. c designates which flip-flop (condition) is to be tested.
2. The condition of the selected flip-flop is unchanged by this instruction.

JUMP IF CONDITION FALSE **JF(cf) loc**

Op Code: 1c0 (adr)
Operation: if condition false, (adr) ⟶ P

Examines the designated flip-flop. If reset, transfers control to (adr). If set, executes the next sequentially available instruction.

			P+1	P+2
7 6	5 4 3	2 1 0	7 0	7 0
1	c	0	LSP	MSP
Op Code			Address	

1. c designates which flip-flop (condition) is to be tested.
2. The condition of the selected flip-flop is unchanged by this instruction.

SUBROUTINE CALL **CALL loc**

Op Code: 106 (adr)
Operation: P+3 ⟶ Stack, (adr) ⟶ P

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

			P+1	P+2
7 6	5 4 3	2 1 0	7 0	7 0
1	0	6	LSP	MSP
Op Code			Address	

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

SUBROUTINE CALL IF CONDITION TRUE **CT(cf) loc**

Op Code: 1(c+4)2 (adr)

Operation: If condition true, P+3 ⟶ Stack, (adr) ⟶ P
Examines the designated flip-flop. If set, transfers the address of the next sequentially available instruction to the pushdown Stack, and transfers control to (adr). If reset, executes the next sequentially available instruction.

			P+1	P+2
7 6	5 4 3	2 1 0	7 0	7 0
1	4	2	LSP	MSP
Op Code			Address	

1. c designates which flip-flop (condition) is to be tested.
2. The condition of the selected flip-flop is unchanged by this instruction.
3. The Stack is open-ended in operation. If it is overfilled, the deepest address will be lost.

SUBROUTINE CALL IF CONDITION FALSE **CF(cf) loc**

Op Code: 1c2 (adr)
Operation: If condition false, P+3 ⟶ Stack, (adr) ⟶ P

Examines the designated flip-flop. If reset, transfers the address of the next sequentially available instruction to the pushdown Stack, and transfers control to (adr). If set, executes the next sequentially available instruction.

			P+1	P+2
7 6	5 4 3	2 1 0	7 0	7 0
1	c	2	LSP	MSP
Op Code			Address	

1. c designates which flip-flop (condition) is to be tested.
2. The condition of the selected flip-flop is unchanged by this instruction.
3. The Stack is open-ended in operation. If it is overfilled, the deepest address will be lost.

SUBROUTINE RETURN RET

Op Code: 007

Operation: (Stack) $\longrightarrow$ P

Transfers control to the address specified by the most recent entry into the pushdown Stack. Deletes the most recent entry from the Stack.

7 6	5 4 3	2 1 0
0	0	7

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

SUBROUTINE RETURN IF CONDITION TRUE RT(cf)

Op Code: 0 (c+4) 3

Operation: If condition true, (Stack) $\longrightarrow$ P.

Examines the designated flip-flop. If set, transfers control to the address specified by the most recent entry into the pushdown Stack and deletes the most recent entry into the Stack. If reset, executes the next sequentially available instruction.

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

1. c designates which flip-flop (condition) is to be tested.
2. The condition of the selected flip-flop is unchanged by this instruction.
3. The effect of attempting more RETURN instructions than the Stack is capable of handling is undefined.

SUBROUTINE RETURN IF CONDITION FALSE RF(cf)

Op Code: 0c3

Operation: If condition false, (Stack) $\longrightarrow$ P

Examines the designated flip-flop. If reset, transfers control to the address specified by the most recent entry into the pushdown Stack and deletes the most recent entry into the Stack. If set, executes the next sequentially available instruction.

7 6	5 4 3	2 1 0
0	c	3

1. c designates which flip-flop (condition) is to be tested.
2. The condition of the selected flip-flop is unchanged by this instruction.
3. The effect of attempting more RETURN instructions than the Stack is capable of handling is undefined.

SHIFT RIGHT CIRCULAR SRC

Op Code: 012

Operation: $A_{(N)} \longrightarrow A_{(N-1)}$; $A0 \longrightarrow A7$, $A0 \longrightarrow$ Carry

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.

7 6	5 4 3	2 1 0
0	1	2

The Zero, Parity and Sign flip-flops are not affected by this instruction.

SHIFT LEFT CIRCULAR SLC

Op Code: 002

Operation: $A_{(N-1)} \longrightarrow A_{(N)}$; $A_7 \longrightarrow A_0$ $A_7 \longrightarrow$ Carry

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.

7 6	5 4 3	2 1 0
0	0	2

The Zero, Parity and Sign flip-flops are not affected by this instruction.

NO OPERATION NOP

Op Code: 300

Operation: P+1 $\longrightarrow$ P

No operation is performed

7 6	5 4 3	2 1 0
3	0	0

The Zero, Parity and Sign flip-flops are not affected by this instruction.

HALT HALT

Op Code: 000, 001, or 377

Timing: Execution stops

Operation: The processor halts

When the START button on the console is depressed, operation resumes at P+1.

If USER mode is set this instruction will cause a privileged instruction interrupt to occur.

POP — POP

Op Code: 060
Operation: (Stack)⟶H,L

Transfers the most recent Stack entry into the H & L registers. H=MSP, L=LSP

7 6	5 4 3	2 1 0
0	6	0

PUSH — PUSH

Op Code: 070
Operation: H,L⟶Stack

Transfers the contents of the H & L registers into the pushdown Stack. H=MSP, L=LSP.

7 6	5 4 3	2 1 0
0	7	0

INPUT — INPUT

Op Code: 101
Operation: (I/O Bus)⟶A

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

7 6	5 4 3	2 1 0
1	0	1

Priv. Note: If USER mode is set this instruction will cause a privileged instruction interrupt to occur.

ENABLE INTERRUPTS — EI

Op Code: 050

Following the next instruction, EI will allow the interrupts to occur until a DISABLE INTERRUPT instruction is executed.

7 6	5 4 3	2 1 0
0	5	0

Priv. Note: If USER mode is set this instruction will cause a privileged instruction interrupt to occur.

DISABLE INTERRUPTS — DI

Op Code: 040

Prevents interrupts from occurring until an ENABLE INTERRUPT instruction is executed.

7 6	5 4 3	2 1 0
0	4	0

Priv. Note: If USER mode is set this instruction will cause a privileged instruction interrupt to occur.

SELECT ALPHA MODE — ALPHA

Op Code: 030

Selects the ALPHA MODE registers and control flip-flops.

7 6	5 4 3	2 1 0
0	3	0

Priv. Note: If USER mode is set this instruction will cause a privileged instruction interrupt to occur.

SELECT BETA MODE — BETA

Op Code: 020

Selects the BETA MODE registers and control flip-flops.

7 6	5 4 3	2 1 0
0	2	0

Priv. Note: If USER mode is set this instruction will cause a privileged instruction interrupt to occur.

EXTERNAL COMMAND — EX (exp)

Op Code: 121 to 153

Operation: Performs I/O control according to (exp)

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, and cassette decks are explained in the sections covering these devices.

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

Table 5-1 is a list of the External Commands. For a detailed discussion of their use, reference should be made to Part 6 (Input/Output Operations) and to descriptions of the separate external devices. External Commands 155-177 are not listed, as they apply to systems with integral cassette units and are described in Part 4 (Cassette Tapes).

Priv. Note: If USER mode is set this instruction will cause a privileged instruction interrupt to occur.

TABLE 5-1
EXTERNAL COMMANDS

EX (exp)

(exp)	OCTAL CODE	COMMAND	DESCRIPTION	DEVICE ADDRESS
ADR	121	Address	Selects device specified by A register	ALL
STATUS	123	Sense Status	Connects selected device status to input lines	↑
DATA	125	Sense Data	Connects selected device data to input lines	│
WRITE	127	Write Strobe	Signals selected device that output data word is on output lines	│
COM1	131	Command 1	Outputs a control function to selected device	│
COM2	133	Command 2	Outputs a control function to selected device	│
COM3	135	Command 3	Outputs a control function to selected device	↓
COM4	137	Command 4	Outputs a control function to selected device	ALL
BEEP	151	Beep	Activates tone producing mechanism	ALL
CLICK	153	Click	Activates audible click producing mechanism	ALL

5.8.4 Category 2 — Augmented Category 1 Instructions

LOAD REGISTER FROM MEMORY USING BC, DE, OR XA FOR THE ADDRESS **L(rd)M (rp)**

Op Code: rp 3d7
Operation: (M)$\longrightarrow$(rd),d $\leq$ 6
Length: 2 bytes
Example: LEM BC

Identical to the L(rd)M instruction except that the specified register pair, instead of HL, is used for the memory address.

LOAD MEMORY FROM REGISTER USING BC, DE, OR XA FOR THE ADDRESS **LM(rs) (rp)**

Op Code: rp 37s
Operation: (rs)$\longrightarrow$ M , s$\leq$6
Length: 2 bytes
Example: LMB DE

Identical to the LM(rd) instruction except that the specified register pair, instead of HL, is used for the memory address.

ARITHMETIC AND LOGICAL OPERATIONS TO OTHER THAN THE A REGISTER

Mnemonics:	Examples:
(op)(rs) (r)	ADAB adds A to B
(op)M (r)	ADMC adds (HL) to C
(op)(r) (vvv)	SUC 20 subtracts 20 from C
SRC (r)	SRCB shifts B right
SLC (r)	SLCD shifts D left

Op Codes: r 2ps, r 0p7, r 0p4, r 012, r 002
Timing: Add 1.0 to equivalent category 1 instruction timing.
Length: Add 1 byte to the equivalent category 1 instruction.

Identical to the equivalent category 1 arithmetic operations except that the specified register, instead of the A register, is used as the accumulator.

SHIFT RIGHT EXTENDED **SRE, SRE(r)**

For SRE:
Op Code: 032
Operation: $A_N \longrightarrow A_{(N-1)}$ Carry$\longrightarrow A_7$, $A_\phi \longrightarrow$ Carry
Length: 1 byte

For SRE(r): Op Code: r 032
Operation: $(r)_N \rightarrow (r)_{(N-1)}$ Carry $\rightarrow (r)_7, (r)_0 \rightarrow$ Carry
Length: 2 bytes

The register is shifted right one place with the left hand bit being replaced by the Carry and the Carry being replaced by the right-hand bit.

I/O USING OTHER THAN THE A REGISTER **IN(r), EX(rs) (exp)**

For IN(r): Op Code: r 101
Operation: (I/O Bus) $\rightarrow$ (r)
Length: 2 bytes
For EX (rs) (exp): Op Code: r 121, r 123, etc.

Operation: Performs I/O control with the specified register according to (exp)
Length: 2 bytes

Identical to the 2200 I/O operations except that the specified register, instead of the A register, is used.

PARITY CHECKING INPUT **PIN, PIN(r)**

For PIN: Op Code: 103
Length: 1 byte
For PIN (r): Op Code: r 103
Length: 2 bytes

Identical to the INPUT instruction except that if the nine bits of the I/O Bus contain an even number of ones, an interrupt will occur.

PUSH USING BC, DE, OR XA **PUSH (rp)**

Op Code: rp 070
Operation: (rp) $\rightarrow$ Stack
Length: 2 bytes

Pushes the specified register pair onto the Stack.

PUSH IMMEDIATE **PUSH loc**

Op Code: 051 (adr)
Operation: (adr) $\rightarrow$ Stack
Length: 3 bytes

Pushes the value of the operand onto the Stack.

POP USING BC, DE, OR XA **POP(rp)**

Op Code: rp 060
Operation: (Stack) $\rightarrow$ (rp)
Length: 2 bytes

Pops the Stack into the specified register pair.

5.8.5 Category 3 — Multi-byte (string) Operations

BLOCK TRANSFER OR BLOCK TRANSFER REVERSE **BT, BTR**

For BT: Op Code: 021

Length: 1 byte
For BTR: Op Code: 111 021

Length: 2 bytes

The Block Transfer instructions move the number of bytes specified in the C register from the field pointed to by HL to the field pointed to by DE while adding the contents of the A register to each byte transferred. BT causes the pointers to be incremented after each transfer while BTR causes the pointers to be decremented after each transfer. If the B register is not zero, the transfer will stop if a character which is equal to the 2's complement of the B register is stored in the destination field (stops after the matching character is moved).

Entry:	HL=location of first source byte.
	DE=location of first destination byte.
	C=number of bytes to move (C=1 to 255; 0 for 256).
	B=2's complement of terminating character if not 0.
	A=8-bit value added to each byte as it is moved (for de-zoning and zoning decimal numbers).
Exit:	HL=location past last source byte.
	DE=location past last destination byte.
	A=entry value.
	B=entry value.
	C=zero or count before terminator character found.
	Condition flags are all altered.
Stack:	1 entry used.
Caution:	Since BT and BTR instructions can take up to 609 microseconds to execute, care must be exercised in their use if time critical interrupt driven programs are to be simultaneously executed.

BLOCK CONVERT **BCV**

Op Code: 062 021

Length: 2 bytes

BLOCK CONVERT is a variation of BLOCK TRANSFER, where the field pointed to by the DE registers is translated byte-by-byte using the translate table pointed to by the HL registers.

Entry: HL=location of the translate table (must not cross a page boundary).
DE=location of the first byte to be translated.
C=number of bytes to move
B=2's complement of terminating character if not 0.
A=no entry value used.

Exit: HL=undefined
DE=location past last destination byte
A=LSB of last table position used for translation.
B=entry value.
C=zero or count before termination character found.

Algorithm: 1. Get the byte pointed to by DE.
2. Set A to the sum of the byte added to L.
3. Get the byte pointed to by HA. This is the table's translated byte.
4. Store the translated byte where DE points
5. Increment DE.
6. B is added to the translated byte.
7. Stop if the Carry and Zero conditions are true — a match is found.
8. Decrement the C register. (Add -1)*
9. Go to Step 1 if result is non-zero.

Stack: 1 entry used

Caution: Since BCV instructions can take over 840 microseconds to execute, care must be taken in their use if time critical interrupt driven programs are to be simultaneously executed.

* A decrement operation is actually an add of -1.

BINARY FIELD ADD WITH CARRY OR SUBTRACT WITH BORROW BFAC, BFSB

For BFAC: Op Code: 011
Length: 1 byte
For BFSB: Op Code: 031
Length: 1 byte

These instructions take the field pointed to by HL and either add it to or subtract it from the field pointed to by DE, leaving the result in the field pointed by DE. The fields may be 1 through 16 bytes in length.

Entry: HL=location of right hand byte of the operand field.
DE=location of right hand byte of the accumulator field
C=the field width (1 through 16; 0 or 16 implies 16).
Carry=carry or borrow into the operation.

Exit: HL=location to left of the left hand byte of the operand field.
DE=location to left of the left hand byte of the Accumulator field.
C=indeterminate.
Carry=carry or borrow out of the operation (all the condition flags are altered).

Algorithm: 1. Load the implicit register from C.
2. Get the byte pointed to by HL.
3. Add it with carry or subtract it with borrow from the byte pointed to by DE and store the result where DE points.
4. Decrement HL and DE by one.
5. Decrement the implicit register by one.
6. Go to step 2 if the implicit register is not now zero.

Stack: 1 entry used

BLOCK COMPARE BCP

Op Code: 041
Length: 1 byte

This instruction matches two strings of bytes from left to right until either a mismatch is found or the specified maximum number of bytes have been scanned.

Entry: HL=location of left hand byte of the subtracting field.
DE=location of left hand byte of the subtracted from field.
C=the maximum number of bytes to scan (1 thru 255; 0 implies 256).

Exit: IF A MISMATCH WAS FOUND:
HL=location after the mismatch in the subtracting field
DE=location after the mismatch in the subtracted from field
C=entry value minus number of bytes that matched
Condition flags all reflect the result of the subtract instruction that found the two bytes differing.
IF ALL BYTES MATCHED
HL=location after the last byte in the subtracting field
DE=location after the last byte in the subtracted from field
C=zero
Condition flags are indeterminate. (Zero condition being set true)

Algorithm: 1. Get the byte pointed to by HL.
2. Subtract the byte pointed to by DE from it.
3. Increment DE and HL.

4. Exit if the Zero condition is false.
5. Decrement C. (Add -1)
6. Go to Step 1 if C is not equal to zero.
7. Exit with the Zero condition true.

Stack: 1 entry used.

DECIMAL FIELD ADD WITH CARRY **DFAC**

Op Code: 111 041

Length: 2 bytes.

This instruction takes the field of zoned BCD digits pointed to by HL and adds it to the field of zoned decimal digits pointed to by DE, leaving the result in the field pointed to by DE. The zone bits of the result field are set to the zone bits in the B register. The fields may be 1 through 16 bytes in length.

Entry: Same as for the BFAC instruction except B=output zoning (right 4 bits must be 0; left 4 bits must be other than 0000).

Exit: Same as for the BFAC instruction except A register is destroyed. B=entry value.

Algorithm:
1. Load the implicit register from C.
2. Get the byte pointed to by HL.
3. Add it with carry to the byte pointed to by DE.
4. Strip away the zone bits.
5. Clear the Carry and go to step 7 if the result is less than 10.
6. Subtract 10 from the result and set the Carry.
7. Set the zoning bits.
8. Store the result where DE points.
9. Decrement HL and DE by one.
10. Decrement the implicit register by one.
11. Go to step 2 if the implicit register is not zero.

Stack: 1 entry used.

NOTE: The binary values for the zoned BCD digits with xxxx not equal to 0000 are as follows (the digits are not packed, i.e., only one digit per byte):

0:xxxx0000	5:xxxx0101
1:xxxx0001	6:xxxx0110
2:xxxx0010	7:xxxx0111
3:xxxx0011	8:xxxx1000
4:xxxx0100	9:xxxx1001

DECIMAL FIELD SUBTRACT WITH BORROW **DFSB**

Op Code: 062 041

Length: 1 byte

This instruction takes the field of zoned BCD digits pointed to by HL and substracts it from the field of zoned BCD digits pointed to be DE, leaving the result in the field pointed to by DE. The zone bits of the two fields must be identical. The zone bits of the result field are set to the zone bits in the B register. The fields may be 1 through 16 bytes in length.

Entry: same as for the DFAC instruction.

Exit: same as for the DFAC instruction.

Algorithm:
1. Load the implicit register from C.
2. Get the byte pointed to by HL.
3. Subtract it, with borrow, from the byte pointed to by DE.
4. Go to Step 6 and clear the Carry if the byte result is not negative.
5. Add 10 to the result and set the Carry.
6. Set the zone bits to those in the B register.
7. Store the result where DE points.
8. Decrement HL and DE by one.
9. Decrement the implicit register by one.
10. Go to Step 2 if the implicit register is not zero.

Stack: 1 entry used.

BINARY FIELD SHIFT LEFT **BFSL**

Op Code: 075

Length: 1 byte

This instruction shifts a field of bytes in memory left one bit position as if all of the bytes made up one continuous word.

Entry: HL=location of right-hand byte of the field.
C=the field width (1 through 16; 0 or 16 implies 16).
Carry=bit shifted in on right

Exit: HL=location left of the left-hand byte of the field.
C=indeterminate.
A=indeterminate.
Carry=bit shifted out on the left.
All other flags are indeterminate.

Stack: 1 entry used.

BINARY FIELD SHIFT RIGHT **BFSR**

Op Code: 1i1 075

Length: 2 bytes

This instruction is similar to BFSL except the shift is in the opposite direction.

Entry: HL=location of left-hand byte of the field.
C=the field width (1 through 16; 0 or 16 implies 16)
Carry=bit shifted in on left.

Exit: HL=location right of the right-hand byte of the field.
C=indeterminate.
A=indeterminate.
Carry=bit shifted out on the right.
All other flags are indeterminate.

Stack: 1 entry used.

MULTIPLE INPUT **MIN**

Op Code: 111 061
Length: 2 bytes

This instruction moves the number of bytes specified in the C register from a buffered input device to the field pointed to by H&L. The number of bytes moved is the number in the C register modulo 16. To make transferring up to 256 bytes easy yet interruptable, the full eight bit value of the C register is retained during loop counting and exit is made with the C register containing its entry value minus the number of bytes transferred, HL containing its entry value plus the number of bytes transferred, and the Zero condition code reflecting the eight bit result of the last decrementation of the C register. Thus the interruptable loop for transferring the number of bytes indicated by the eight bit value in the C register yet not inhibiting interrupts more than 155 microseconds would appear as follows:

```
LOOP    LA      DEVADR
        DI
        EX      ADR
        EX      DATA
        EI
        MIN
        JFZ     LOOP
```

Note that the device must be re-addressed for each execution of the MIN instruction if an interrupt could cause some other device to be addressed. The MIN instruction causes a parity checking input strobe to be executed every 8 microseconds. This execution operates without regard to any status bits of any kind. There is no existing 2200 system I/O device capable of using this instruction and it is included for use with system I/O devices with parity generation and faster buffers allowing them to be used at data rates equivalent to DMA channels. The MIN instruction has all of the advantages of a non-I/O device interrupting system (lower software overhead in high throughput situations, superior control over the occurrence of events allowing probability of correctness in the program logic and repeatability of event occurence, and simpler hardware using lower speeds and noise filtered buses) and yet achieves DMA throughput rates.

Entry: HL=location of first destination byte
C=number of bytes to move (this number is taken modulo 16 and if it is 0 modulo 16 then 16 bytes will be moved).

Exit: HL=location of entry value plus number of bytes moved
C=entry value minus number of bytes moved

Algorithm:
1. Execute a parity checking INPUT.
2. Store the byte where HL points.
3. Increment HL.
4. Load the implicit register from C.
5. Decrement C using the ALU. (Add -1)
6. Decrement the implicit register.
7. Exit if the implicit register is zero.
8. Decrement the P-counter.
9. Re-fetch the instruction without allowing interrupts.

Stack: 1 entry used.

NOTE: To input a block of 256 bytes using the loop described above would take 2495 microseconds if no interrupts occurred (an average of 9.75 microseconds per byte).

MULTIPLE OUTPUT **MOUT**

Op Code: 111 071
Length: 2 bytes

This instruction is similar to the MIN instruction except for the direction of information flow. MOUT moves the number of bytes specified in the C register from the field pointed to by HL to a buffered output device. A byte is written using the EX WRITE strobe every 8 microseconds and interrupts can be inhibited for a maximum of 155 microseconds. As with MIN there is no existing 2200 system I/O device capable of being used with the MOUT instruction.

NOTE: To output a block of 256 bytes using a loop similar to the one described for MIN (a MOUT instruction would appear where a MIN instruction appears in the example) would take 2495 microseconds if no interrupts occurred (an average of 9.75 microseconds per byte).

5.8.6 Category 4 — Processor State Save and Restore Instructions

STACK STORE **STKS**

Op Code: 065
Length: 1 byte

The STACK STORE instruction POPs a specified number of Stack entries and stores them (LSB followed by MSB) in the field pointed to by HL. Upon entry, HL points to the left-hand byte.

Entry: HL=first location in the storage area
C=the number of entries to be POPPED and stored (1 through 16; 0 or 16 implies 16)

Exit: HL and C indeterminate
Condition flags unchanged

STACK LOAD **STKL**

Op Code: 111 065
Length: 2 bytes

The STACK LOAD instruction pushes onto the Stack the specified number of entries from the field pointed to by HL. Upon entry HL points to the right hand byte and the entries are loaded in reverse order to allow restoring the Stack from locations stored using the STKS instruction.

Entry: HL=last location in the storage area
C=the number of entries to be PUSHED (1 through 16; 0 or 16 implies 16)

Exit: HL=indeterminate
C=indeterminate
Condition flags unchanged

REGISTER STORE **REGS**

Op Code: 055
Length: 1 byte

The REGISTER STORE instruction stores all of the registers for the currently selected mode (ALPHA or BETA) in the field pointed to by the top entry of the Stack. This entry points to the right-hand byte of the field and the registers are stored in reverse order moving from right to left. When the instruction terminates, the top entry of the Stack points to the left of the left-hand byte in the field. For example, if entry is made with the top entry of the Stack pointing to location 02007 (octal), the registers are stored as follows:

02000:A
02001:B
02002:C
02003:D
02004:E
02005:H
02006:L
02007:X

In the above example, the top entry of the Stack will be 01777 when the instruction terminates. The contents of neither the registers nor the condition flags for the given mode are altered by this instruction.

REGISTER LOAD **REGL**

Op Code: 111 055
Length: 2 bytes

The REGISTER LOAD instruction loads all of the registers for "the currently selected" mode (ALPHA or BETA) from the field pointed to by HL. Upon entry, HL points to the right-hand byte of the field. The registers are loaded in reverse order moving to the left in the field. In this manner, the registers can be reloaded from values stored by the REGS instruction. In the example given for the REGS instruction, if the REGL instruction were entered with HL=02007, the registers shown would be loaded from the locations shown. The condition flags are not altered by this instruction.

CONDITION CODE SAVE **CCS, CCS(r)**

Op Code: 042, r 042
Length: 1 byte or 2 bytes if r specified.

This instruction loads the register (r) with a value such that if the value is added to itself using the AD(r) operation, the condition flags will all be restored to their state before the CCS instruction was executed. The logic equations for the value loaded into (r) are:

A7=Carry
A6=Sign
A5=A4=A3=A2=0
A1=Not Zero and Not Sign
A0=Not Zero and Not Parity

This instruction does not alter the state of any of the condition flags. If (r) is not specified, the A register is used.

5.8.7 Category 5 — Address Manipulation Instructions

INCREMENT REGISTER PAIR **INCP**

Mnemonics	Op Codes
INCP HL	015
INCP HL, 2	117 015
INCP HL,A	017
INCP BC	062 015
INCP BC,2	113 015
INCP BC,A	062 017
INCP DE	174 015
INCP DE,2	115 015
INCP DE,A	174 017
INCP XA	022 015
INCP XA,2	111 015
INCP XA,A	022 017

These instructions increment the indicated register pair by either one, two or the contents of the A register. The increment value is added to the LSP register and then the carry is added to the MSP register, if necessary. The A register is not changed, except in the XA case. Other condition flags are indeterminate.

DECREMENT REGISTER PAIR **DECP**

Mnemonics	Op Codes
DECP HL	035
DECP HL,2	117 035
DECP HL,A	037
DECP BC	062 035
DECP BC,2	113 035
DECP BC,A	062 037
DECP DE	174 035
DECP DE,2	115 035
DECP DE,A	174 037
DECP XA	022 035
DECP XA,2	111 035
DECP XA,A	022 037

These instructions decrement the indicated register pair by either one, two, or the contents of the A register. The decrement value is subtracted from the LSP register and then the borrow is subtracted from the MSP register, if necessary. The A register is not changed, except in the case of XA.

DOUBLE LOAD **DL**

Mnemonics	Op Codes
DL DE,HL	047
DL BC,HL	111 047

DL BC,BC	062 047
DL BC,DE	113 047
DL DE,BC	174 047
DL DE,DE	115 047
DL HL,BC	176 047
DL HL,DE	117 047
DL HL,HL	057

These instructions load the register pair specified by the first operand from the memory location pointed to by the register pair specified by the second operand. The LSP register (C, E, or L) is loaded from the specified memory location and the MSP register (B,D, or H) is loaded from the next higher memory location. Note that indirect addressing can be accomplished by loading a register pair from the locations that the pair specify (DL HL,HL for example).

DOUBLE STORE — DS

Mnemonics	Op Codes
DS DE,HL	027
DS BC,HL	111 027
DS BC,DE	113 027
DS DE,BC	174 027
DS HL,BC	176 027
DS HL,DE	117 027

These instructions store the register pair specified by the first operand into the memory locations pointed to by the register pair specified by the second operand. The LSP register (C,E, or L) is stored in the specified memory location and the MSP register (B,D or H) is stored in the next higher location.

PAGED LOAD — PL

Mnemonics	Op Codes
PL A,(loc)	105 LSP
PL B,(loc)	114 LSP
PL C,(loc)	124 LSP
PL D,(loc)	134 LSP
PL E,(loc)	144 LSP
PL H,(loc)	154 LSP
PL L,(loc)	164 LSP

These instructions load the specified register from the memory location specified by the LSP given in the instruction and the MSP in the X register.

PAGED STORE — PS

Mnemonics	Op Codes
PS A,(loc)	107 LSP
PS B,(loc)	116 LSP
PS C,(loc)	126 LSP
PS D,(loc)	136 LSP
PS E,(loc)	146 LSP
PS H,(loc)	156 LSP
PS L,(loc)	166 LSP

These instructions store the specified register in the memory location specified by the LSP given in the instruction and the MSP given in the X register.

DOUBLE PAGED LOAD — DPL

Mnemonics	Op Codes
DPL BC,(loc)	111 124 LSP
DPL DE,(loc)	113 144 LSP
DPL HL,(loc)	115 164 LSP

These instructions load the specified register pair from the memory locations specified by the LSP given in the instruction and the MSP given in the X register. The C,E, or L register is loaded from the specified memory location and the B,D, or H register is loaded from the next higher location.

DOUBLE PAGED STORE — DPS

Mnemonics	Op Codes
DPS BC,(loc)	111 126 LSP
DPS DE,(loc)	113 146 LSP
DPS HL,(loc)	115 166 LSP

These instructions store the specified register pair in the locations specified by the LSP given in the instruction and the MSP given in the X register. The C, E or L register is stored in the specified location and the B, D or H register is stored in the next higher location.

INCREMENT AND DECREMENT INDEX — INCI, DECI

Mnemonics	Op Codes
INCI (disp), (index)	005 LSP(i)
DECI (disp), (index)	025 LSP(i)
INCI*(disp), (index)	111 005 LSP MSP(i)
DECI*(disp),(index)	111 025 LSP MSP(i)

The processor has a construct called an index which is a 16-bit value kept in memory. The concept is similar to index registers except that all the values are kept in the page of memory pointed to by the X register. The index is specified by a single byte in the instructions (shown as (i) above) which points to the memory location containing the LSP of the index value, the MSP being in the next higher memory location ((i) specifies the LSP of the index address while the X register specifies the MSP of the index address). The instruction also contains a displacement (shown as (disp) above) that is either one or two bytes in length (depending upon the op code). These instructions either increment or decrement the value of the index by the displacement. The Carry condition flag reflects the carry or borrow from the incrementation or decrementation. The rest of the condition flags are indeterminate.

Stack: 1 entry used

LOAD FROM INDEX INCREMENTED OR DECREMENTED — LFII, LFID

Mnemonics	Op Codes
LFII BC,(disp), (index)	062 005 LSP(i)
LFID BC,(disp),(index)	062 025 LSP(i)
LFII BC,*(disp),(index)	113 005 LSP MSP(i)
LFID BC,*(disp),(index)	113 025 LSP MSP(i)
LFII DE,(disp),(index)	174 005 LSP(i)
LFID DE,(disp),(index)	174 025 LSP(i)

LFII DE,*(disp),(index)	115 005 LSP MSP(i)
LFID DE,*(disp),(index)	115 025 LSP MSP(i)
LFII HL,(disp),(index)	176 005 LSP(i)
LFID HL,(disp),(index)	176 025 ISP(i)
LFII HL,*(disp),(index)	117 005 LSP MSP(i)
LFID HL,*(disp),(index)	117 025 LSP MSP(i)

These instructions are similar to the INCI and DECI instructions except that they load the specified pair of registers with the result of adding or subtracting the displacement to or from the value of the index. The condition flags are similarly affected.
Stack: 1 entry used.

5.8.8 Category 6 - Operating System Control

BASE REGISTER LOAD **BRL, BRL(r)**
Op Code: 072, r 072

Length: 1 or 2 if r specified

This instruction loads the base register from the specified register. Note that the base register cannot be read. For this reason, loading the base register will normally be a monitor function, allowing the monitor to keep within itself the value of the base register for user state storage purposes. This instruction will cause a privileged instruction interrupt if the USER mode flag is set. If (r) is not specified, the A register is used.

NOP JUMP **NOJ loc**
Op Code: 045 (adr)
P+3 $\rightarrow$ P
Length: 3 bytes.

This instruction increments the P-counter twice. It is useful for overstoring jump instructions which might be executed while being overstored. The procedure to overstore a jump instruction would be to first overstore the op code with an 045 (NOP JUMP) and then update the address portion. Then the op code could be overstored with the appropriate jump instruction. The primary use of this instruction is for overstoring the interrupt vector jump instructions for the interrupts which cannot be disabled (such as MEMORY PARITY FAULT) and which might happen while the jump is being overstored. No condition flags or registers are modified.

SYSTEM CALL **SC**
Op Code: 067

This instruction causes the USER mode flag to be cleared, the last entry in the sector table to be set to the last 4K section of addressable memory space with access protection, and a CALL to be performed to location 0167452 (in the ROM). This is the mechanism via which the user would communicate with an operating system that used the USER mode.

USER RETURN **UR**
Op Code: 111 102

This instruction is identical to the RETURN instruction (op code 007) except that additionally the USER mode flag is set.

SECTOR TABLE LOAD **STL**
Op Code: 077

Length: 1 byte

This instruction loads up to the first 15 entries in the sector table. This table contains eight bits for each entry. Bit 1 is not used and should always be set to zero. Bit 2 is set for access enable. Bit 3 is set for write enable. The left-hand four bits and Bit 0 are used to map that entry into a particular 4K section of *physical* memory space. This instruction will cause a privileged instruction interrupt if the USER mode flag is set.

Entry:	HL=location of first byte in table of up to 15 to load. C=number of entries to load (0 to 15).
Exit:	No registers or condition flags are changed.
Stack:	1 entry used.

BREAKPOINT **BP**
Op Code: 052

Length: 1 byte

This instruction is similar to a SYSTEM CALL (SC) instruction except the call is performed to location 0167460 of system RAM. This will cause entry into the system DEBUG routine if the memory vector is not changed.

ENABLE INTERRUPTS AND JUMP **EJMP (loc)**
Op Code: 111 050 (adr)

Length: 4 bytes

This instruction is identical to the ENABLE INTERRUPTS (EI) instruction except that additionally a jump is performed to the (LSP, MSP) address.

ENABLE INTERRUPTS AND RETURN **EUR**
Op Code: 062 050

Length: 2 bytes

This instruction is identical to the combination of the ENABLE INTERRUPTS, Set USER Mode Flag and RETURN instructions.

Category 7:
5.8.9 6600 Instruction Set

The following is a description of the instructions that are new with the 6600 processor.

DOUBLE PAGED LOAD REVERSED DPLR (rp), loc

Mnemonic	Opcode
DPLR BC, loc	062 114 LSP
DPLR DE, loc	174 134 LSP
DPLR HL, loc	176 154 LSP

Timing: 3.80

These instructions load the specified register pair from the memory locations specified by the LSP given in the instruction and the MSP given in the X-register. The B, D, or H register is loaded from the specified memory location and the C, E, or L register is loaded from the next higher location. Note that this is similar to the 5500 DPL instruction except the order in which the registers are loaded is reversed.

DOUBLE PAGED STORE REVERSED DPSR (rp), loc

Mnemonic	Opcode
DPSR BC, loc	062 116 LSP
DPSR DE, loc	174 136 LSP
DPSR HL, loc	176 156 LSP

Timing: 3.80

These instructions store the specified register pair into the locations specified by the LSP given in the instruction and the MSP given in the X-register. The B, D, or H register is stored into the specified memory location and the C, E, or L register is stored into the next higher location. Note that this is similar to the 5500 DPS instruction except the order in which the registers are stored is reversed.

SECTOR TABLE LOAD STARTING AT OFFSET STLO (r)

Mnemonic	Opcode
STLOA	022 077
STLOB	111 077
STLOC	062 077
STLOD	113 077
STLOE	174 077

Timing: 3.70 + C * 1.25

The Sector Table in the 6600 contains eight bits for each entry. Bit 0 of a Sector Table entry is explained later. Bit 1 of a Sector Table entry contains a hardware generated and checked parity bit. Any value loaded into this bit position is ignored since the hardware generates the proper parity bit when a Sector Table entry is loaded. If, during any memory access, there is not the correct number of one bits out of the eight Sector Table entry bit positions, a Sector Table Parity Error System Call interrupt will be generated to memory location 0167474. Bit 2 of a Sector Table entry is set to enable the sector to be accessed (read or written) when the machine is in User Mode. Bit 3 of a Sector Table entry is set to enable the sector to be written in either User or System Mode. Bits 4 through 7 of a Sector Table entry are used for physical memory address bits 12 through 15 and bit 0 is used for physical memory address bit 16 (giving the 6600 17 bits of physical memory address to accommodate the 128K of physical memory space).

The STLO(r) instruction is similar to the 5500 STL instruction except that the upper four bits of the specified register (A, B, C, D, or E) determine where in the Sector Table the loading is started (the lower four bits can be any value). For example, if the STLOA instruction is performed with the A-register containing a 060 (octal) and the C-register containing a 5, Sector Table entries 3 through 7 will be loaded. Note that if the upper four bits of the specified register plus the lower four bits of the C-register total more than 15, loading will wrap around in the Sector Table into the lower entries and the value that was to be loaded into the top entry will be ignored (the top entry always points to the system ROM sector at 0170000 through 0177777, is access enabled, and not write enabled). For example, if the STLOB instruction is performed with the B-register containing a 0300 and the C-register containing a 7, the Sector Table entries loaded will be 12, 13, 14, 15 (ignored), 0, 1, and 2.

Entry:	HL =	location of first byte in a table of up to 15 Sector Table entries to be loaded
	C =	number of entries to be loaded (0 thru 15; the upper 4 bits of C can be any value)
	(r) =	starting Sector Table entry (upper four bits 0 thru 15; the lower 4 bits of (r) can be any value; (r) can be A, B, D, or E)

Exit: Sector Table loaded
No registers or condition flags changed
1 stack level used

SYSTEM INFORMATION INFO

Opcode: 111 010
Timing: 2.50

This instruction is used to differentiate the 6600 from other Datapoint processors. In the 5500, this instruction performs no operation. In the 6600, this instruction loads a 1 into the A-register and the revision number of the micro-code ROM into the B-register. None of the condition code flags and none of the other registers are affected by this instruction. To determine the type of Datapoint processor in which the program is running, the following sequence is suggested:

```
XRA                  Determine if 2200
LLA
LHA
DECP    HL           (This is a NOP on a 2200)
JFC     IMA2200      I'm a 2200
XRA                  Determine if 5500 or 6600
INFO
ORA
JTZ     IMA5500      I'm a 5500
JMP     IMA6600      I'm a 6600
```

BINARY FIELD LEFT TO RIGHT OPERATIONS

BFLR(op)

Mnemonic	Opcode
BFLRAD	111 006
BFLRAC	111 016
BFLRSU	111 026
BFLRSB	111 036
BFLRND	111 046
BFLRXR	111 056
BFLROR	111 066

Timing: 6.30 + N * 2.15

These instructions are similar to the 5500 BFAC and BFSB instructions except that the memory pointers are incremented after each operation is performed (instead of being decremented). In addition, logical and non-carry operators are allowed (the non-carry operators are useful for incrementing a number of one-byte counters appearing in contiguous memory).

DOUBLE MEMORY TO REGISTER OPERATIONS

D(op)M (rp)

Mnemonic	Opcode
DADM (rp)	I rp I 013
DACM (rp)	I rp I 310
DSUM (rp)	I rp I 033
DSBM (rp)	I rp I 330
DNDM (rp)	I rp I 043
DXRM (rp)	I rp I 053
DORM (rp)	I rp I 063
DCPM (rp)	I rp I 073

Timing: 4.60 to 5.65

These instructions perform the indicated operation between the 16-bit value at the memory location pointed to by the HL register pair (LSB at the location pointed to and MSB at the next higher location) and the 16-bit value in the specified register pair (BC, DE, HL, or XA). In subtraction and comparison, the value in memory is subtracted from the value in the register pair, and in all operations except comparison the result is deposited in the register pair. The Carry, Sign, and Zero condition flags reflect the entire 16-bit result (the Carry is always set false by the logical operations) while the Parity condition flag is undefined after the operation.

DOUBLE PAGED TO REGISTER OPERATIONS

D(op)P (rp),loc

Mnemonic	Opcode
DADP (rp), loc	I rp+1 I 013 LOCLSB
DACP (rp), loc	I rp+1 I 310 LOCLSB
DSUP (rp), loc	I rp+1 I 033 LOCLSB
DSBP (rp), loc	I rp+1 I 330 LOCLSB
DNDP (rp), loc	I rp+1 I 043 LOCLSB
DXRP (rp), loc	I rp+1 I 053 LOCLSB
DORP (rp), loc	I rp+1 I 063 LOCLSB
DCPP (rp), loc	I rp+1 I 073 LOCLSB

Timing: 5.15 to 6.20

These instructions are similar to the D(op)M instructions except that the memory locations used are pointed to by the memory address contained in the instruction (LSP) and the X-register (MSP).

DOUBLE IMMEDIATE TO REGISTER OPERATIONS

D(op)I (rp), data

Mnemonic	Opcode
DADI (rp), data	I rp I 110 LSB MSB
DACI (rp), data	I rp I 311 LSB MSB
DSUI (rp), data	I rp I 130 LSB MSB
DSBI (rp), data	I rp I 331 LSB MSB
DNDI (rp), data	I rp I 140 LSB MSB
DXRI (rp), data	I rp I 150 LSB MSB
DORI (rp), data	I rp I 160 LSB MSB
DCPI (rp), data	I rp I 170 LSB MSB

Timing: 4.00 to 5.05

These instructions are similar to the D(op)M instructions except that the operand data is actually the last two bytes in the instruction.

DOUBLE REGISTER TO MEMORY OPERATIONS

DM(op) (rp)

Mnemonic	Opcode
DMAD (rp)	I rp+1 I 110
DMAC (rp)	I rp+1 I 311
DMSU (rp)	I rp+1 I 130
DMSB (rp)	I rp+1 I 331
DMND (rp)	I rp+1 I 140
DMXR (rp)	I rp+1 I 150
DMOR (rp)	I rp+1 I 160

Timing: 5.30 to 6.35

These instructions are similar to the D(op)M instructions except that the direction of data flow is reversed. On subtraction, the register pair value is subtracted from the memory locations, and in all operations the result is deposited into the memory locations (specified by the HL register pair). Note that there is no comparison operation.

SINGLE PAGED TO REGISTER OPERATIONS

P(op) (r), loc

Mnemonic	Opcode
PAD (r), loc	I r I 106 LOCLSB
PAC (r), loc	I r I 112 LOCLSB
PSU (r), loc	I r I 122 LOCLSB
PSB (r), loc	I r I 132 LOCLSB
PND (r), loc	I r I 142 LOCLSB
PXR (r), loc	I r I 152 LOCLSB
POR (r), loc	I r I 162 LOCLSB
PCP (r), loc	I r I 172 LOCLSB

Timing: 3.40 (except 3.25 for CP)

These instructions perform the indicated operation between the 8-bit value in the memory location specified by the last byte in the instruction (LSB) and the X-register (MSB) and the 8-bit value in the specified register with all, except the comparison operation, depositing the result in the specified register. All condition flags are set to reflect the result as in an (op) (r) operation.

DOUBLY LINKED LIST DELETE **LLDEL**

Opcode: 111 051
Timing: 9.40

A doubly linked list construct in the 6600 appears as follows:

ITEM1	DA	ITEM2	forward pointer
	DA	ITEM3	backward pointer
ITEM2	DA	ITEM3	forward pointer
	DA	ITEM1	backward pointer
ITEM3	DA	ITEM1	forward pointer
	DA	ITEM2	backward pointer
ITEM4	DA	00000	item to be inserted
	DA	00000	

When the linked list delete instruction is performed with HL pointing to ITEM2, the instruction deletes ITEM2 from the list by moving its forward pointer to the forward pointer of ITEM1 and its backward pointer to the backward pointer of ITEM3. When the instruction completes, the entry value of HL has not been changed while the DE register is left pointing to ITEM1 and BC is left pointing to ITEM3. None of the condition flags are changed by this instruction (ITEM4 will be referrenced in the following description).

DOUBLY LINKED LIST INSERT **LLINS**

Opcode: 062 051
Timing: 10.80

This instruction inserts a list item into a linked list construct. Using the example shown for the LLDEL instruction, if the insert instruction is performed with DE pointing to ITEM2 and HL pointing to ITEM4, the instruction exits with ITEM2's forward pointer pointing to ITEM4, ITEM4's forward pointer pointing to ITEM3, ITEM3's backward pointer pointing to ITEM4, and ITEM4's backward pointer pointing to ITEM2. Finally, the entry values of the DE and HL registers are unchanged and the BC register is left pointing to ITEM3. None of the condition flags are changed by this instruction.

INTEGER MULTIPLY: **IMULT**
HLDE = HL * BC

Opcode: 111 011
Timing: If H = 0: 26.20 + N * 2.00
If H # 0: 45.55 + N * 2.00
(N = number of 1's in HL)

This instruction multiplies the unsigned values in HL and BC putting an unsigned result in the HLDE register quadruple (most significant byte in H and least significant byte in E). When the instruction completes, the Zero condition flag reflects the 16-bit result in the HL register pair and the Carry condition flag is set if the sign bit of the D register is a one. The A, B, C, and X registers are not changed by this instruction. The Sign and Parity condition flags are undefined.

DOUBLE INTEGER DIVIDE:
HLDE/BC = > Q(DE),R(HL) **DIDIV**

Opcode: 111 031
Timing: If error: 3.55
Else: 57.40 to 82.20

This instruction produces an error indication with the Carry condition flag set if the BC register pair is less than or equal to the HL register pair (in unsigned arithmetic). Otherwise, it divides the unsigned HLDE register quadruple by the BC register pair placing the quotient in the DE register pair and the remainder in the HL register pair. Upon completion of the instruction, the Carry condition is left cleared to indicate that an error did not occur and the Zero condition is set based upon the 16-bit value in the HL register pair. The A, B, C, and X registers are unchanged by this instruction. The Sign and Parity condition flags are undefined.

INTEGER DIVIDE: **IDIV**
DE/BC = > Q(DE),R(HL)

Opcode: 062 031
Timing: If error: 3.90
Else: 57.75 to 82.55

This instruction is identical to the DIDIV instruction except that the HL registers are first loaded with zero.

2'S COMPLEMENT A REGISTER PAIR **COMP (rp)**

Mnemonic	Opcode
COMP BC	062 011
COMP DE	174 011
COMP HL	176 011

Timing: 4.75 if complement
3.70 otherwise

If the sign bit of the A-register is set, this instruction performs a 2's complement upon the specified register pair. Upon completion of this instruction, only the specified register pair's contents can be changed and the condition flags are undefined.

2'S COMPLEMENT A REGISTER PAIR **COMPS (rp)**

Mnemonic	Opcode
COMPS BC	113 011
COMPS DE	115 011
COMPS HL	117 011

Timing: 5.20 if complement
4.15 otherwise

This instruction is identical to the COMP instruction except that the sign bit of the A-register is duplicated in the next lower A-register bit position.

5.8.10 Instruction Timing

The following table shows the 5500 and 6600 timings for those instructions that are implemented in the 5500 processor.

Instruction	5500 timing	6600 timing
L(rd)M	2.60	1.75
L(rd)M (rp)	3.40	2.60
LM(rd)	2.60	1.75
LM(rd) (rp)	3.40	2.60
L(rd) (rs)	1.20	1.00
L(r) data	1.80	1.45
AD(rs)	1.40	1.15
AC(rs)	1.40	1.15
SU(rs)	1.40	1.15
SB(rs)	1.40	1.15
ND(rs)	1.40	1.15
XR(rs)	1.40	1.15
OR(rs)	1.40	1.15
CP(rs)	1.20	1.00
AD(rs) (rd)	2.40	2.00
AC(rs) (rd)	2.40	2.00
SU(rs) (rd)	2.40	2.00
SB(rs) (rd)	2.40	2.00
ND(rs) (rd)	2.40	2.00
XR(rs) (rd)	2.40	2.00
OR(rs) (rd)	2.40	2.00
CP(rs) (rd)	2.20	1.85
ADM	2.60	2.10
ACM	2.60	2.10
SUM	2.60	2.10
SBM	2.60	2.10
NDM	2.60	2.10
XRM	2.60	2.10
ORM	2.60	2.10
CPM	2.40	1.95
ADM(rd)	3.60	2.95
ACM(rd)	3.60	2.95
SUM(rd)	3.60	2.95
SBM(rd)	3.60	2.95
NDM(rd)	3.60	2.95
XRM(rd)	3.60	2.95
ORM(rd)	3.60	2.95
CPM(rd)	3.40	2.80
AD data	2.20	1.60
AC data	2.20	1.60
SU data	2.20	1.60
SB data	2.20	1.60
ND data	2.20	1.60
XR data	2.20	1.60
OR data	2.20	1.60
CP data	2.00	1.45
AD(r) data	3.20	2.45
AC(r) data	3.20	2.45
SU(r) data	3.20	2.45
SB(r) data	3.20	2.45
ND(r) data	3.20	2.45
XR(r) data	3.20	2.45
OR(r) data	3.20	2.45
CP(r) data	3.00	2.30
SLC	1.40	1.15
SRC	1.40	1.15
SRE	1.40	1.15
SLC(r)	2.40	2.00
SRC(r)	2.40	2.00
SRE(r)	2.40	2.00
JMP loc	2.60	2.05
Jcc loc	2.80	2.25
Jcc loc (fall thru)	1.40	1.10
EJMP loc	4.40	3.40
NOJ loc	1.40	1.00
NOP	1.20	0.70
CALL loc	2.80	2.20
Ccc loc	3.20	2.45
Ccc loc (fall thru)	1.60	1.20
RET	1.80	1.30
Rcc	2.00	1.50
Rcc (fall thru)	1.00	0.80
UR	3.20	2.45
EUR	3.80	3.15
IN	5.00	5.00
IN(r)	6.00	5.85
PIN	5.40	5.00
PIN(r)	6.40	5.85
EX (exp)	9.20	7.00
EX(r) (exp)	10.20	7.85
MIN	3.00 + N * 8.80	2.95 + N * 8.30
MOUT	3.00 + N * 8.40	2.95 + N * 8.30
BETA	1.40	1.20
ALPHA	1.40	1.20
DI	1.40	1.20
EI	1.40	1.20
POP	2.20	1.45
POP (rp)	3.00	2.30
PUSH	1.80	1.15
PUSH (rp)	2.60	2.00
PUSH loc	2.60	2.05
BT (A=B=0)	4.80 + N * 3.20	6.70 + N * 1.60
BT (A, B≠0)	4.80 + N * 3.40	6.00 + N * 2.35
BTR (A=B=0)	5.80 + N * 3.60	7.85 + N * 1.60
BTR (A, B≠0)	5.80 + N * 3.80	6.95 + N * 2.35
BCV (A=B=0)	5.80 + N * 4.80	7.55 + N * 2.50
BCV (A, B≠0)	5.80 + N * 5.00	6.85 + N * 3.25
BCP (if match)	5.20 + N * 2.60	5.35 + N * 1.95
BCP (mismatch)	4.40 + N * 2.60	4.85 + N * 1.95
BFAC	5.00 + N * 2.80	5.35 + N * 2.15
BFSB	5.00 + N * 2.80	5.35 + N * 2.15
DFAC	5.40 + N * 4.50	6.20 + N * 3.45
DFSB	5.40 + N * 3.80	6.30 + N * 2.90
BFSL	3.80 + N * 2.20	3.00 + N * 1.70
BFSR	4.60 + N * 2.00	3.40 + N * 1.55

STKS	1.60 + N * 2.40	1.55 + N * 1.70
STKL	4.40 + N * 2.20	3.60 + N * 1.70
REGS	13.00	9.10
REGL	12.20	9.85
CCS	2.40 to 3.00	1.65 to 2.45
INCP HL	2.80	1.40 or 1.95
INCP HL,A	3.00	1.55 or 2.10
INCP (rp)	3.60	2.45 or 3.00
INCP (rp),2	3.80	2.50 or 3.05
INCP (rp),A	3.80	2.40 or 2.95
DECP HL	2.80	1.40 or 1.95
DECP HL,A	3.00	1.55 or 2.10
DECP (rp)	3.60	2.45 or 3.00
DECP (rp),2	3.80	2.50 or 3.05
DECP (rp),A	3.80	2.40 or 2.95
DL DE,HL	3.60	2.50
DL BC,HL	5.40	3.95
DL BC,BC	4.80	3.75
DL BC,DE	5.20	3.90
DL DE,BC	4.80	3.75
DL DE,DE	5.20	3.90
DL HL,BC	4.80	3.75
DL HL,DE	5.20	3.90
DL HL,HL	3.60	2.50
DS DE,HL	3.60	2.50
DS BC,HL	5.40	3.95
DS BC,DE	5.20	3.90
DS DE,BC	4.80	3.75
DS HL,BC	4.80	3.75
DS HL,DE	5.20	3.90
PL (r),loc	3.00	2.20
PS (r),loc	3.00	2.20
DPL (rp), loc	5.00	3.80
DPS (rp),loc	5.00	3.80
INCI (dsp),(idx)	7.40	3.65 or 5.10
DECI (dsp,(idx)	7.60	3.65 or 5.10
INCI *(dsp),(idx)	9.40	7.25
DECI *(dsp),(idx)	9.60	7.45
LFII (rp),(dsp),(idx)	7.40	5.60
LFID (rp),(dsp),(idx)	7.60	5.80
LFII (rp),*(dsp),(idx)	8.40	6.25
LFID (rp),*(dsp),(idx)	8.60	6.45
BRL	1.20	1.00
BRL(r)	2.20	1.85
STL	3.20 + N * 1.80	2.70 + N * 1.25
SC	2.00	1.80
BP	2.20	2.00

APPENDIX A
SYSTEM ROM OPERATING DESCRIPTION

CHAPTER 1. SYSTEM ROM FUNCTIONS

1.1 INTRODUCTION

The Datapoint 6600 ROM occupies 4K of *physical* memory. (From 0170000 to 0177777). Four major routines are executed in the ROM with which the user should be familiar. They are POWERUP, RESTART, DEBUG, and MEMORY TEST.

1.2 POWERUP

The first major ROM routine, POWERUP, is executed when the 6600 is (initially) supplied with power. This routine disables the one millisecond interrupt, selects ALPHA Mode, writes zeroes in all of RAM Memory to initialize memory parity (note that there is no machine state to save), and calls a subroutine SETUP which does six things:

(1) Loads the Sector Table entries 0 => 016 (0 => 14 decimal) with values to make a one-for-one translation from based logical space to physical space with no protection set. (Note that the 017th entry in the Sector Table is always set to point to the 4K sector of physical memory (0170000 =>0177777) with USER and WRITE access disabled.)

(2) Clears the User Mode Flag.

(3) Initializes the Base Register to zero.

(4) Loads a partial character set in the RAM display.

(5) Clears all entries in the Breakpoint Table (which is also in System RAM).

(6) Initializes the Interrupt Vector Table in System RAM (to the internal trap messages).

The vectors are loaded as indicated in the following RAM memory locations:

0167400	MEMORY PARITY FAILURE VECTOR.
0167406	INPUT PARITY FAILURE VECTOR.
0167414	OUTPUT PARITY FAILURE VECTOR.
0167422	WRITE PROTECT VIOLATION VECTOR.
0167430	ACCESS PROTECT VIOLATION VECTOR.
0167436	PRIVELEDGED INSTR VIOLATION VECTOR.
0167444	ONE MILLISECOND CLOCK VECTOR.
0167452	USER SYSTEM CALL VECTOR.
0167460	BREAKPOINT VECTOR.
0167466	UNASSIGNED INSTRUCTION
0167474	SECTOR TABLE PARITY ERROR

Note that the ONE MILLISECOND INTERRUPT is disabled during the time any System interrupt is executed. Under the normal ROM initialization sequence, the ONE MILLISECOND INTERRUPT vector is pointed back into the ROM where the ONE MILLISECOND INTERRUPTS are re-enabled prior to the jump to location zero. If a user program alters any of the SYSTEM INTERRUPT VECTORS, it must also be responsible for re-enabling interrupts (EI) if required.

System RAM is the term used to denote the 256-byte page of RAM Memory (From 0167400 to 0167777) which contains Interrupt Vector Locations in the first 128 bytes and the Machine State Storage Area and the Diagnostic Scratch Area in the second 128 bytes.

Interrupts are generated in 6600 firmware through the 'System-Call" mechanism which shifts program execution into 6600 ROM locations which contain JMP's to Interrupt Vectors in the System RAM.

The Interrupt Vectors consist of six byte entries to enable Vector Address Modification through the use of the NOJ instruction. (See NOJ description in Sec. 5.8.8.)

The POWERUP sequence concludes by loading the RAM display with an abbreviated ASCII character set (all unloaded characters are set to triangles) and HALTING to invoke the bootstrap mechanism.

The POWERUP routine contains an operating feature which gives the user the capability of moving the logical sector of memory which contains the System RAM to the bottom (zeroth) physical sector (on RAM card 1) and moving the rest of the memory up one sector in physical memory. This feature could conceivably be of use in the case where the System RAM memory failed and the user wanted to get into DEBUG to run the memory test (particularly if the memory failure was intermittent). To do this the KEYBOARD and DISPLAY Keys must BOTH be depressed at the time of POWERUP.

1.3 RESTART

The second major ROM routine, RESTART, is invoked by momentarily depressing the RESTART and RUN keys, by the machine being halted (by other than the STOP key) when either a cassette is in place in the rear deck with the right-hand tab punched out, or when no cassette is in place in the rear deck and the head gate is closed. Note that if the DISPLAY key is depressed at the time RESTART is invoked, the Diagnostic routine (DEBUG) will be entered.

The RESTART routine disables the one millisecond interrupt, puts the 6600 in ALPHA Mode, and checks for diagnostic activation (DISPLAY key depressed). If DEBUG is not selected for execution, RESTART calls SETUP and then executes the bootstrap function which will load a block of data from a cassette tape in the rear deck or from a disk peripheral that contains a disk that is on line.

If the rear cassette deck has a cassette tape in place, the tape will be rewound and the first block of data read into low RAM and then executed.

Otherwise, disk peripheral devices are scanned in the order of Mass Storage (I/O addresses 0113, 0115, and 0116), Cartridge Disk (I/O address 0170), and Diskette

(I/O addresses 074, 072, 071) for a disk on line in drive zero. If no such disk is found, the cassette deck is checked again. If during this loop the operator depresses the DISPLAY key, the drive number checked in each of the disk peripherals is incremented once each time through the loop. If the drive number is incremented from 255, a click is sounded and the drive number is returned to zero. If a disk is found on line, a 256 byte block is read (from cylinder 0, head 0, sector 3 of Mass Storage; cylinder 0, head 0, sector 3 of Cartridge Disk; track 0, sector 4 of Flexible Disk). If the format of this block is correct, it is loaded into memory and executed. If the format is not correct, the same action is taken as if the disk is not on line., A click is sounded for every sector read from a disk peripheral.

The format of the block of data on the disk is L H -L -H (252 bytes of data). The L is the LSB and the H is the MSB of the address of where the data is to be loaded. The -L and -H are the 1's complements of the L and H values. The first byte of the data must be a zero and will be overstored with the drive number from which the block was loaded. Execution is begun at the location of the second byte of data. User programs may cause a Restart to occur by jumping to the Restart routine entry point at 0170033 in the ROM.

CHAPTER 2. DEBUG

2.1 INTRODUCTION

The Datapoint 6600 DEBUG is a ROM-resident program whose immediate accessibility creates a flexible interface between user and machine. This guide is intended to provide the 6600 user with that information essential to the use of the ROM-DEBUG System Test.

2.2 STARTUP PROCEDURE

There are four methods of entry to DEBUG:

(1) Forcing entry through manual intervention.

(2) Entry through a BREAKPOINT set by DEBUG.

(3) Entry through a BREAKPOINT imbedded in the user program.

(4) Entry as the consequence of a RETURN from a DEBUG Call Command.

TO FORCE ENTRY INTO DEBUG:

DEPRESS DISPLAY, RUN, RESTART; keeping each key depressed until all three are down.

Then release RUN or RESTART.

This will bring up the DEBUG display and commands may be entered.

2.3 SAVING THE MACHINE STATE

When DEBUG is entered through console intervention, most of the user's program state is undisturbed. What is not saved is the state of the interrupt enable flip-flop (interrupts are disabled), the state of the base register or sector table (these two are not changed upon entry to DEBUG), the state of ALPHA/BETA Mode flip-flop (all registers are saved), the state of the I/O system (what device is addressed and the state of its status/data selection flip-flop), and the bottom two Stack locations.

What is saved are the ALPHA/BETA Mode registers and condition code flip-flops, and the 14 Stack entries (the top entry containing the P-counter).

Note that there exist default values upon exit from DEBUG for:

(1) ALPHA/BETA Mode flip-flop.

(2) Currently addressed device and its Status/Data Mode flip-flop.

These can be changed using DEBUG commands ('A', 'G', and 'R').

2.4 DISPLAY FORMAT

The 6600 DEBUG display consists of five lines and occupies the bottom-right corner of the screen.

LLLLLL	: LOGICAL ADDRESS (IF ORIGIN NOT 0)
BBBBBB	: PHYSICAL ADDRESS
* NNN	: ASCII, 8 BIT OCTAL C[CURADR]
MMMMMM	: LSB, MSB ADDRESS FORMED AT CURADR.
nnnnnnn*	: COMMAND INTERPRETER

The first (top) line shows the *logical* address only if Origin (See O command) is non-zero.

The second line shows the current *physical* (Based) sixteen bit address, referred to as CURADR below.

The third line contains both an ASCII (One character shown as *) and an 8-bit octal (Three characters shown as NNN) representation of the contents of the current *physical* address byte.

The fourth line contains an octal representation of the 16-bit value whose LSB is at CURADR and whose MSB is at CURADR+1. (This is the address format used by JMP, CALL, and DA mnemonics).

THE COMMAND INTERPRETER

The bottom line of the display is used to edit and input commands to DEBUG. The blinking cursor signifies that the Command Interpreter is awaiting user input.

Data is entered serially into the input display buffer. The cursor is displaced to the right successively as this occurs. The BACKSPACE Key erases the character most recently entered, shifting the entry cursor to the left one space. The CANCEL Key deletes the entire entry.

All commands are single characters. Commands which accept input arguments are preceded by the argument which is entered in octal. Not all commands require an input argument. The last character input to the interpreter must be a legal command. Illegal input is ignored, evoking a BEEP from the 6600.Commands are executed upon their entry into the interpreter (no ENTER Key is required and the command character is not displayed), with the current contents of the entry line being cleared. Upon command completion the cursor reappears, awaiting further input.

2.5 COMMAND SYNTAX

This explanation of the command syntax uses the following notation:

nnn	Indicates an optional sequence of octal digits not to exceed the number of n's given.
(nnn) nnn	If input argument contains more than eight bits of significance, special results will occur. In general what will happen is

that two bytes of memory will be affected by the command, either a register pair or a memory address in LSB, MSB format.

nnnnnn 16-bit argument. No digits usually causes special action.

12345 There exists a set of special commands whose accidental execution is inhibited by the requirement that they contain this unique argument.

2.6 INPUT COMMAND LIST

nnnA Address the given or current (if nnn not given) I/O device. The current I/O device is the last one selected by this command. No check is made on address format. STATUS is displayed as C[CURADR]. Note that the current device is readdressed and put into the mode last accessed (Data mode if 'F' or 'G' have been executed subsequent to last 'A' command) prior to resuming execution through CALL, EXECUTE, JUMP or USER RETURN Commands.

nnnnnnB Store a BREAKPOINT instruction at the given or current address. Upon BP execution the state of the machine is saved, the memory location at which the BP was set is restored to its original value and the corresponding BP table entry is cleared.

The following notes reference the use of the 'B' command.

Overlay BREAKPOINT will not loop. That is: It is not possible to successfully set a BREAKPOINT in the same memory location in order to iterate the execution of a program loop. To iterate BREAKPOINT through a looping sequence requires 'double BREAKPOINTING'.

Ten BREAKPOINTS can be active at any one time. Note that BP's DISABLE interrupts and leave them disabled prior to resuming execution through CALL, EXECUTE, JUMP or USER RETURN commands. This is done to enable testing of foreground routines with DEBUG. (If it becomes necessary to use DEBUG with interrupts enabled, the user can enable interrupts on return with the "i" command.) Note that it is impossible for the machine to determine its current register (ALPH/BETA) mode. Therefore the 'R' command mode flip-flop is set to ALPHA when a BP is encountered. If the user wishes to test code written in BETA Mode it is necessary that he manually put the 6600 in BETA Mode (with the 'R' command) prior to resumption of execution through CALL, EXECUTE, JUMP or USER RETURN commands. Similarly, the USER may have to address the proper I/O device (with A) and perhaps put it into DATA Mode (with G) before continuing execution from a BREAKPOINT. Note that DEBUG will not set a BREAKPOINT over another BREAKPOINT.

nnnnnnC Call the given or current address. The Machine State is restored before execution control is passed to the Subroutine. A RETURN from the Called Subroutine causes re-entry into DEBUG and hence, for the Machine State to again be saved.

nnnnnnD Decrement the current address value by one or value (nnnnnn).

nnnnnnE Continue execution from a forced or BREAKPOINT entry into DEBUG. Machine State is restored prior to resumption of execution. The interrupts are left disabled. The register mode is set to the last R value (initialized to ALPHA Mode upon BP or forced entry), the base register and sector table are not changed, and the I/O device is addressed and optionally set to DATA Mode. If a new execution address is given (n), the top Stack location will be changed to (n) prior to continuation of execution.

nnnF Fetch next data byte from current or given I/O device. The F Command will automatically put device in DATA Mode and the device will subsequently be put in DATA mode when the E command is given.

nnnG Go to DATA mode in the current or given I/O device when the E command is given.

H Not used

nnnnnnI Increment the current address value by one or value (nnnnnn).

nnnnnnJ Jump to the given or current address. Machine State is restored prior to resumption of execution.

12345K Set ASCII keyin mode. Will allow ASCII data to be entered into CURADR in auto-increment mode (i.e. will update CURADR). BACKSPACE moves CURADR back and displays its contents. DELete moves CURADR forward and displays its contents. CANCEL causes a return to normal mode.

L	Link to the address pointed to by the Current Address. CURADR is replaced by line 3 (the 16-bit LSB, MSB address formed at CURADR, CURADR+1). The remaining display parameters are updated appropriately. Note that initial display state upon entry into DEBUG can be regenerated by performing the 'S' command, followed immediately by the 'L' command.
(nnn) nnnM	Modify the contents of the current address location. If the value of the Input Argument exceeds eight significant bits, two memory locations will be modified, treating the input argument as an address in LSB, MSB Format for JMP and DA. (A CLICK is sounded to notify the operator if an MSB is stored).
nnnnnnN	Set physical address to nnnnnn.
O	Origin mode (useful for debugging relocatable code) — performs the following four functions. (Utilizes upper and lower case O).

1.	O	clears Origin mode.
2.	nnO	Sets Origin table pointer. The Origin table is 10 entries deep and entries of 0-011 are valid.
3.	o	Sets addressing bias to selected table entry (2 above)
4.	nnnnnno	Modifies selected table entry to (n) and sets address bias to that value.
	NOTE:	Setting Origin mode also displays top address line (Logical Address).

nnnnnnP	Load the Base Register with the 8-bit value = (nnnnnn - 0100000)>8
12345Q	Load the Sector Table. CURADR => Table whose first byte equals the number of entries to be loaded. The following bytes contain arguments to be loaded into the Sector Table.
R	Switch ALPHA/BETA Mode register display. The ASCII character displayed after command execution tells the current display mode: A=ALPHA, B=BETA.
nnS	Display the specified Stack item (up to 015 Octal). Note: Entry into DEBUG pushed P onto the top of the Stack.
12345T	Start primary memory test. Displays Memory Size and Pass Counter in right-bottom corner of screen. Maintains running display of test failures.
nnnnnnU	User mode execute with optional return to (n) address. Command sets USER mode and then executes i Command. (Interrupts enabled)

nnnV	EX COM4	The I/O device must be addressed with A command.
nnnW	EX WRITE	STATUS is displayed.
nnnX	EX COM1	after the command is issued.
nnnY	EX COM2	'nnn' is the current output byte.
nnnZ	EX COM3	The previous nnn value is used if none is given.

?	Displays the processor version, the revision level of the Micro-Code and the revision level of the Macro-Code.

SHIFTED COMMAND CHARACTERS

nnn x	Display	'X'	register or modify to (nnn)
(nnn) nnn a		'A'	modify register pair if input
nnn b		'B'	argument exceeds eight bits
(nnn) nnn c		'C'	
nnn d		'D'	the LSB register specifies the
(nnn)nnn e		'E'	pair (i.e. L for H & L)
nnn h		'H'	
(nnn) nnn l		'L'	

nnn f	Displays or modifies the condition flag byte. Flag bits: 7=>C; 6=>S; 1=> - Z & - S; 0 =>- Z & - P. The bit pattern which displays the condition flags will replicate the previous state when added to itself.
o	See Origin mode.
nnnnnn i	Same as 'E', but with interrupts enabled.
nnnnnn s	PUSH value (n) onto Stack.
nn r	POP Stack (nn) times.
nnn p	Load Base register direct with value (nnn).
12345 t	Start Pseudo-random memory test.
nnn y	EX DATA with (nnn) on output Bus.
nnn z	EX STATUS, with (nnn) on output Bus.
nnnnnnENTER	Set Logical Address (physical if no origin) to nnnnnn. Command has no effect unless it is preceded by an Input Argument.
CANCEL	Cancel entry line.
BACKSPACE	Backspace on entry line.
(nnn) nnn.	Modify the contents and then increment the current address. If Input Argument has more than eight significant bits, two memory locations are modified, treating

	the argument as an address in LSB, MSB Format. (a CLICK is sounded).
(nnn) nnn∧	Modify the contents and then increment the current address. If Input Argument is null, the last non-null value given is used. If 'last value' exceeded eight bits of significance, two memory locations will be modified. (a CLICK is sounded).
#	Clear all active (DEBUG set) breakpoints, restoring values.

6600 ROM DEBUG COMMAND SUMMARY

nnn A	Address the (n) or current I/O device.	
nnn nnn B	Set a breakpoint to the (n) or current address.	
nnn nnn C	Call the (n) or current address.	
nnn nnn D	Decrement the current address by (n) or 1.	
(nnn nnn) E	Continue execution or replace top stack location with (n) and continue execution.	
nnn F	Fetch next data byte from (n) or current device.	
nnn G	Go to DATA mode in (n) or current device on 'E', 'U' or 'i' command.	
(nnn) nnn I	Increment the current address by (n) or 1.	
nnn nnn J	Jump to the given (n) or current address.	
12345 K	Set ASCII keyin mode.	
L	Link to address pointed to by current address.	
(nnn) nnn M	Modify the contents of the current address.	
nnnnnn N	Set physical address to nnnnnn.	
nn O	Select Origin table entry.	
[*] (ENTER)	Set Origin addressing to entry value and display.	
[*] (nnn)	Set Origin addressing to (n), enter in table, and display.	
(nnn) nnn P	Load Base register with (nnnnnn - 0100000)>8	
12345 Q	Load the sector table.	
R	Switch ALPHA/BETA mode and display.	
nn S	Display the (Nth) Stack location item.	
12345 T	Start the primary 6600 memory test.	
nnn nnnn U	Continue execution as in 'E' command but in USER mode. (Interrupts enabled)	
nnn V	EX COM4	Device must be addressed for I/O commands.
nn W	EX WRITE	Status is displayed after command issue.
nnn X	EX COM1	'nnn' is the output byte.
nnn Y	EX COM2	
nnn Z	EX COM3	
?	Displays processor version, Micro-Code and Macro-Code revision levels.	

SHIFTED COMMAND CHARACTERS

nnn x	Display X register or modify to (nnn)
(nnn) nnn a	A modify register pair if
nnn b	B argument exceeds eight bits.
(nnn) nnn c	C
nnn d	D The LSB register specifies
(nnn) nnn e	E the pair. (i.e. L for H&L)
nnn h	H
(nnn) nnn l	L
nnn f	Displays or updates the condition flags.
nnn nnn i	Same as 'E' above with interrupts enabled.
nnn nnn h	PUSH value (nnn nnn) onto Stack.
nn s	POP Stack (nn) times.
nnn p	Load Base register direct with value (nnn).
12345 t	Alternate 6600 memory test.
nnn y	EX DATA (nnn) on output bus.
nnn z	EX STATUS (nnn) on output bus.
nnnnnn ENT	Set Logical address to 'nnnnnn'.
CAN	Cancel entry line.
BKSP	Backspace one on entry line.
(nnn) nnn.	Modify and increment.
nnn (nnn)∧	Modify and increment using the last non-null value.

DATAPOINT CORPORATION

The leader in dispersed data processing ™