HEWLETT PACKARD

Hardware Internal Design Specification

For the HP-71

Hewlett-Packard -- Portable Computer Division

Corvallis, Oregon

HP-71 HARDWARE

INTERNAL DESIGN SPECIFICATION

Detailed Design Description

SEPTEMBER 1984

HP Part No. 00071-90071

Table of Contents

SYSTEM OVERVIEW	CHAPTER 1

The HP-71 mainframe chip set is composed of the 4 custom CMOS integrated circuits shown below:

CHIP SET:

```
1  1LF2 CPU
3  1LF3 Display Driver (@ .5K bytes RAM/chip)
1  1LG7 ROM hybrid     (@ 4 chips/hybrid and 16K bytes ROM/chip)
4  1LG8 RAM hybrids    (@ 4 chips/hybrid and  1K bytes RAM/chip)
```

The CPU and display drivers are mounted on the underside of the keyboard/LCD PC board. The ROM hybrid and 4 RAM hybrids are mounted on the I/O PC board. Connection to the 4 RAM/ROM ports in the front of the HP-71 and the HP-IL port in the back of the machine is made via the I/O PCB. The card reader port connects to the keyboard PC board.

The CPU (1LF2) is an arithmetic oriented microprocessor capable of both HEX and BCD arithmetic. The 1LF2 operates on a fully multiplexed 4-bit bus with a 512K byte address space and an internal word size of 64 bits. The CPU has a general purpose Input Register (16 lines) with interrupt capability and a general purpose Output Register (12 lines). The Input and Output Registers are used to form the HP-71 keyboard matrix. The Output Register is also used to drive the daisy-chain signal to each port and to drive the piezo-electric beeper used in the system. External circuitry provides the capability of loud or soft beep.

The three display drivers (1LF3) drive the 8 row by 136 column (including annunciators) display. One of these 3 1LF3s acts as the master. The master generates the display clock, the voltage reference, and the display-on signal used by the other two display chips (slaves). There is a 4-bit writeable register on the master that controls the display contrast by adjusting the value of the voltage reference signal. Each display chip supplies .5K bytes of hard-configured RAM and a 24-bit timer (crystal-oscillator controlled) with 1/512 second resolution. The timers are used by the operating system to implement the clock system. The master display chip also provides a low supply voltage sensor that is used by the operating system to indicate low-battery.

Each 1LG8 chip provides 1K bytes of soft configured RAM. The four 4-chip RAM hybrids in combination with the 1.5K bytes of display driver RAM provide a system total of 17.5K bytes. Plug-in RAM modules contain one 4-chip hybrid totalling 4K bytes of RAM. Each 1LG7 chip provides 16K bytes of ROM. The 64K operating system is stored in one 4-chip ROM hybrid that is hard addressed beginning at 00000. Plug-in ROM modules contain 1 to 4 hard or soft configured ROMs. For information on hard and soft address configuration see section 2.2.

The HP-71 system supports a low-power shutdown mode. The system can be shutdown with the display either on or off. For information on power down or wake up see section 2.4.

The system is powered from either the unregulated battery voltage or the regulated AC adapter voltage. A 470uF capacitor supplies "keep-alive" power to the system while batteries are being replaced with the system in shutdown mode.

```
+-----------------------------------------------+-------------------+
|                                               |                   |
|   BUS COMMUNICATION                           |   CHAPTER  2      |
|                                               |                   |
+-----------------------------------------------+-------------------+
```

This section describes the HP-71 system bus structure, protocol, and timing. Also included is a description of the bus commands, device addressing, and the power down and wake up characteristics.

2.1 Bus Structure

The HP-71 bus consists of 8 lines including:

- 4 BUS[0:3] - data lines, driven by the CPU or system devices.
- 1 *STROBE (*STR) - driven by the CPU.
- 1 *COMMAND-DATA (*CD) - usually driven by the CPU.
- 1 VDD
- 1 GROUND

Plug-ins may have two additional lines:

- 1 DAISYIN - (DIN) input to device
- 1 DAISYOUT - (DOUT) output from device, may be tied to next device's DIN

NOTE : A '*' before a signal name denotes negative true logic.

2.1.1 General Protocol

*STR is driven by the CPU and serves to synchronize all bus transfers. It is not a true system clock since it can remain inactive for several cycles while the CPU is performing internal manipulations. All address, data, and bus commands are transferred on the BUS[0:3]. The BUS[0:3] is driven during *STR low by either the CPU or the device the CPU is accessing. This data is latched either by the CPU during *STR low or by the receiving device on the rising edge of *STR.

All bus operations are initiated by the CPU. The CPU starts a specific transfer on the bus by driving the *CD line low before *STR goes low. While *CD and *STR are low the CPU drives a bus command on the BUS[0:3] and all devices in the system latch the command on the rising edge of *STR. This strobe is referred to as a command strobe. The bus command issued during a command strobe specifies the operation that is to be performed on each succeeding *STR until another bus command is issued. At all times when data or address is being transferred *CD is held high. A strobe issued while *CD is high is referred to as a a data strobe.

2.1.2 Bus Commands

The bus commands are:

0	NOP	All devices ignore *STR until a new command is loaded.
1	ID	The unconfigured device that has its DAISY-IN high sends its 5-nibble ID on the following data strobes, starting with the low- order nibble of the ID.
2	PC READ	(PC)->BUS or read using the Program Counter (PC). The device addressed by its program counter sends data pointed to by its local program counter on each following data strobe and all devices increment their local program counters once each data strobe. A dummy strobe immediately follows the issuance of this bus command (see subsection 2.1.4).
3	DP READ	(DP)->BUS or read using the Data Pointer (DP). The device addressed by its data pointer sends data pointed to by its local data pointer on each following data strobe and all devices increment their local data pointers once each data strobe. A dummy strobe immediately follows the issuance of this bus command (see subsection 2.1.4).
4	PC WRITE	BUS->(PC) or write using PC. The device addressed by its program counter loads the data on the following data strobes into the location pointed to by its local program counter and all devices increment their local program counter once each data strobe.
5	DP WRITE	BUS->(DP) or write using DP. The device addressed by its data pointer loads the data on the following data strobes into the location pointed to by its local data pointer and all devices increment their local data pointer once each data strobe.

6	LOAD PC	BUS->PC or load PC. All devices load the data on following 5 data strobes into their local program counter, starting with the low-order nibble. After all 5 nibbles are transferred the command code is automatically changed to a 2, PC READ (see subsection 2.1.3).
7	LOAD DP	BUS->DP or load DP. All devices load the data on following 5 data strobes into their local data pointer, starting with the low-order nibble. After all 5 nibbles are transferred the command code is automatically changed to a 3, DP READ (see subsection 2.1.3).
8	CONFIGURE	The unconfigured device that has its DAISY-IN high loads the following 5 data nibbles into its configuration register starting with the low-order nibble.
9	UNCONFIGURE	The device currently addressed by its data pointer unconfigures itself. The device then responds to CONFIGURE and ID bus commands only. The local data pointers must be loaded immediately preceding an unconfigure command.
A	POLL	All chips that require service pull one data line high during the next *STR low (see section 2.5).
B	Reserved	
C	BUSCC	The device currently addressed by its local data pointer performs a specific operation as defined by the individual device.
D	Reserved	
E	SHUTDOWN	When the CPU has received a SHUTDN instruction it issues this command and turns off its oscillator. Each device responds based on its own special requirements to this command (see section 2.4).
F	RESET	All devices reset their configuration flags (if applicable) and perform other local resets based on their own special requirements.

2.1.3 Command Auto-Switch

There exists one special case in which all devices change their current bus command. This is called 'auto-switch' and occurs following the load of either the PC or the DP. On the rising edge of *STR after the 5th nibble of address has been loaded all devices clear bit 2 of their command latch changing the bus command from either a LOAD PC to PC READ or LOAD DP to DP READ.

2.1.4 Dummy Strobe

Immediately following a PC READ bus command, a DP READ bus command, and a command auto-switch the CPU issues a 'dummy strobe'. This dummy strobe appears as a data strobe except that no data is transferred during this period and devices do not increment their local address registers. The dummy strobe provides memory devices a full strobe cycle for the first access and therefore allows data pipelining.

2.2 Addressing

Each device on the HP-71 bus has two 20-bit address registers; a local program counter (PC) and a local data pointer (DP). Each device is also either hard addressed at a specific address (hard configured) or capable of being dynamically located within the address space (soft configured). A device only responds to data reads and writes if its local address register (PC or DP depending on the read or write command) is within its configured address space.

The HP-71 operating system allows soft configured devices to have address spaces ranging in size from 8 bytes to 128K bytes. All devices are configured such that the upper-order bits of the local address register can be compared with the upper-order bits of the device's configuration register (hard or soft). If these bits are identical, the device has an address match and will respond to read and write commands (and the unconfigure and BUSCC commands if applicable). Each device with a given address space size compares a given number of the upper-order bits of address. For example, a device with an address space size of 1K bytes or 2K nibbles requires 11 bits of address leaving the upper 9 bits for its configuration register.

2.2.1 Soft Configuration

A soft configured device powers up unconfigured. When unconfigured a device responds only to the ID and CONFIGURE commands and drives its DAISYOUT low. A device's ID code is used to identify the device before it is configured. If a soft configured device is unconfigured and has its DAISYIN line high, it sources its 5-nibble ID code starting with the low-order nibble on the 5 data strobes immediately following the issuance of an ID command (no dummy strobe is issued).

The 5-nibble ID code contains information on the device type and the address space required by the device as defined below:

NIBBLE 0 : Determines the size of the address space and is interpreted differently for memory and memory mapped I/O.

Nib 0	Memory Size	MM I/O space
F	1K nibble	16 nibbles
E	2	32
D	4	64
C	8	128
B	16	256
A	32	512
9	64 (max RAM)	1K
8	128	2K
7	256 (max)	4K
6	-	8K
7	-	16K (max)

NIBBLE 1 : Reserved for future use.

NIBBLE 2 : Device type--

- 0 : RAM
- 1 : ROM
- 2-E : Assorted memory types
- F : Memory-mapped I/O

NIBBLE 3 : Device class--

Memory : unassigned

Memory-mapped I/O - 0 : HP-IL mailbox
1-F : unassigned

NIBBLE 4 :
- bits 0-1 : unassigned
- bit 2 : Last chip in sequence. Always assumed high for MM I/O.
- bit 3 : Last chip in module.

Since the BUS[0:3] is precharged low before each strobe, the CPU will read an ID of all zeros if all devices are configured (or are unconfigured but have DAISYIN low). For more information on how the operating system handles configuration see the HP-71 Software IDS Volume 1.

A soft configured device is assigned its address configuration by the CONFIGURE command. If an unconfigured device has its

DAISYIN line high, it loads the configuration address that is issued on the 5 data strobes immediately following the CONFIGURE command (low-order nibble first) into its configuration register. A device may actually latch only the number of high-order bits it requires as determined by its address space size.

After being configured a device no longer responds to either an ID or CONFIGURE command. A configured device drives DAISYOUT to the same logic level as DAISYIN. The DAISYOUT of one device may be tied to the DAISYIN of second device. In this way many devices may be daisy-chained together in a way that they can be configured one at a time to different addresses. After being configured a device waits until the next command strobe to set its configuration flag in order to delay DAISYOUT so that the next device on the daisy-chain will not be configured simultaneously.

A device may be unconfigured by either a RESET or UNCONFIGURE bus command. The bus RESET command unconfigures all soft configured devices in the system. A device responds to an UNCONFIGURE command by clearing its configuration flag if the DP is within its address configuration.

2.2.2 Hard Configuration

A hard configured device powers-up configured to a specific address. It will not respond to an ID, CONFIGURE, or UNCONFIGURE command and a bus RESET will not affect its configuration. If the device has a DAISYOUT, it is always driven to the same logic level as its DAISYIN.

2.3 Data Transfer

All information that is transferred from the CPU to other devices in the system (commands, addresses, and data) is latched from the BUS[0:3] on the rising edge of *STR. Data that is transferred from system devices to the CPU is latched off the BUS[0:3] after the falling edge of *STR (timed internally on the CPU).

The CPU loads all devices' local address registers by issuing a LOAD PC or LOAD DP bus command followed by 5 data strobes of address, least significant nibble first. After the last nibble of address has been loaded all devices auto-switch to a PC READ or DP READ bus command. The CPU may then read the contents of that address location by issuing one dummy strobe followed by 1 to 16 data strobes during which the CPU latches the BUS[0:3] data. The

CPU may read without first loading the local address registers by issuing a PC READ or DP READ, followed by a dummy strobe, followed by 1 to 16 data strobes. The CPU precharges the BUS[0:3] low each cycle before *STR goes low. Therefore if no device responds the CPU reads zeros.

The CPU writes the contents of a specific addressed location similarly. It is not required to load the local address registers immediately before issuing a PC WRITE or DP WRITE command. The write command is followed by 1 to 16 data strobes during which the addressed device latches the BUS[0:3] data.

All devices increment their local address registers once each data strobe during read and write operations. It is possible for a read or write operation to begin in one device and cross the address boundary into another device. Future controllers on the HP-71 bus may read and write more than 16 nibbles at a time.

Two other types of data transfers are ID, which is simply a 5-nibble read with no address load or dummy strobe, and CONFIGURE, which is a 5-nibble write with no address load. Both these data transfers require that a device be unconfigured and that the DAISYIN line be high. POLL is a unique read and is discussed in section 2.5.

2.4 Power down; Wake up

The HP-71 system can be shutdown under software control. The CPU executes a SHUTDN instruction by issuing a SHUTDOWN bus command and on the ensuing cycle stopping the system clock (*STR) and its own oscillator. While in shutdown mode all data stored in RAM and CPU resident memory is preserved. The CPU is brought out of shutdown mode by either pulling an Input Register line high, or by driving *CD low.

*CD is driven low to wake up the CPU primarily by a device in the system that needs service while the system is in shutdown mode. If a device wakes up the CPU and the CPU shuts down without satisfying its service request the device will not wake up the CPU again until its service request has been satisfied and it needs service again. This avoids a situation where the operating system does not know how to handle a device's service request and cannot shutdown. For more information on power down and wake up see section 3.5.

2.5 Service Poll

If a device needs service while the CPU is operating it must wait until the CPU executes a service request instruction (SREQ), or, if it has the capability, interrupt the CPU using IR14 (available at all ports). The SREQ instruction causes the CPU to issue a POLL bus command followed by one data strobe during which the CPU latches the BUS[0:3] data in the manner of a usual read. A device may respond to the service POLL by pulling one of the BUS[0:3] lines high. Since the CPU precharges the BUS[0:3] low every cycle before *STR goes low the data read by the CPU is a binary OR of all devices' responses.

The following HP-71 chips can wake-up the CPU and can respond to a service POLL on the BUS[0:3] line shown:

Device	Bus line	Reason for Service Request
Display Driver	BUS[0]	Timer underflow.
HP-IL chip	BUS[1]	Data Avail; Interrupt; Pwr-on Reset; Loop Service Request.
Card Reader	BUS[2]	FIFO servicing; Error condition.

PROCESSOR	CHAPTER 3

3.1 CPU Overview

The 1LF2 CPU is a proprietary CPU optimized for high-accuracy BCD math and low power consumption. The CPU's principle function is to fetch and execute micro-instructions in proper sequence. Memory is accessed in 4-bit quantities called "nibbles". Addresses are 20 bits, providing a physical address space of 512K bytes. The CPU operates internally on four 4-bit busses with one or two busses acting as the source and one or two busses acting as the destination. Data for either source bus can originate from the system Bus, CPU resident memory, or other CPU register. Operations performed by the ALU include "add", "subtract", "and", "or", "1's complement", "2's complement", and in combination with the shifter "bit-shift", "digit-shift", and "rotate", on up to 64-bit operands.

For information regarding the full CPU instruction set see Chapter 4, entitled "HP-71 ASSEMBLER INSTRUCTION SET".

3.2 Pin Designations

The 1LF2 CPU's external pins are as follows:

PIN	FUNCTION
VDD	Power supply.
GND	System ground potential.
BUS[0:3]	System bus.
*STR	*STROBE
*CD	*COMMAND-DATA

IR[0:15]	INPUT REGISTER - For keyboard input and general input.
OR[0:11]	OUTPUT REGISTER - For keyboard output and general output.
HALT	HALT - When asserted high, the CPU completes the current instruction, tristates the Bus, *STR, and *CD lines and waits in a loop until HALT goes low again. The Bus is held passively low, *STR and *CD passively high.
NIF	NEXT INSTRUCTION FETCH - Goes high to indicate occurrence of the next instruction following the current *STR.
*INT	INTERRUPT - When pulled low the CPU completes the current instruction, pushes the PC onto the top of the subroutine return stack, and initiates the interrupt routine.
OSC1,OSC2	Oscillator input/output used for LC connection.
ECE	EXTERNAL CLOCK ENABLE - tied low on PC board.
CIO	CLOCK I/O - The primary oscillator frequency is driven out on this pin.
DRI	DRIVE - High when the CPU is driving the Bus, low if the CPU is sensing or tristated.
ENP	ENABLE POWER SUPPLY - Tied low on PC board.
VCO	VOLTAGE CONTROL OUT - Not used.
VCI	VOLTAGE CONTROL IN - Not used.

Note that *INT is available at all ports except the Card Reader port; IR14 is available at all ports; and HALT is available at all ports except PORT1. The Output Register is not only used to form the keyboard matrix, but also drives the DAISYIN (DIN) signal to each port and the piezo-electric beeper.

3.3 Registers

There are two types of registers on the CPU; those used for data transfers and arithmetic operations, and those used for program and system control.

Arithmetic Registers:

A	64-bits	Working register - I/O register
B	64-bits	Working register
C	64-bits	Working register - I/O register
D	64-bits	Working register
R0	64-bits	Scratch register
R1	64-bits	Scratch register
R2	64-bits	Scratch register
R3	64-bits	Scratch register
R4	64-bits	Scratch register
CARRY	1-bit	Flag set by arithmetic operations and tests

Control Registers:

POINTER	4-bits	Pointer register
DPOINT0	20-bits	Address pointer register
DPOINT1	20-bits	Address pointer register
PC	20-bits	Program Counter
RETURN STACK	20-bits	8-level subroutine stack
STATUS	16-bits	Program status flags
HW. STATUS	4-bits	CPU/system status flags
OUTPUT	12-bits	Keyscan/write only Output Register
INPUT	16-bits	Keyscan/read only Input Register

3.3.1 Working and Scratch Registers

All arithmetic operations are performed using the 4 working registers: A, B, C, and D. Data transfers are performed principally with the A and C registers.

The scratch registers are used to temporarily hold the contents of the working registers. The lower 20 bits of scratch register R4 are reserved by the operating system for interrupt processing, and therefore are not normally available for data storage.

3.3.1.1 Field Selection

Subfields of the working registers may be manipulated using field selection. The possible field selections range from the entire register to any single nibble of the register. Certain subfields are designed for use in BCD calculations and others are designed for use in general data manipulation and data access.

FIELD SELECTION FIELDS

P	Digit pointed to by P register
WP	Digit 0 through digit pointed at by P
XS	Digit 2 - Exponent sign
X	Digits 0-2 - Exponent and exponent sign
S	Digit 15 - Mantissa sign
M	Digits 3-14 - Mantissa
B	Digits 0-1 - Exponent or byte field
W	Digits 0-15 - Whole word
A	Digits 0-4 - Address field

Nibbles of Register

```
-----------------------------------------------
15:14:13:12:11:10: 9: 8: 7: 6: 5: 4: 3: 2: 1: 0
-----------------------------------------------
|S|                                   |XS|<-B->|
                                |<---- A ----->|
  |<-------------- M ---------------->|<-- X ->|
|<--------------------- W -------------------->|
```

3.3.2 Carry Bit

The Carry bit is adjusted when a arithmetic operation or test is performed. During a calculation, such as incrementing or decrementing a register, it is set if the calculation overflows or borrows and cleared if it does not. During a arithmetic test, such as comparing two registers for equality, it is set if the test is true and cleared if it is not.

3.3.3 Pointer Registers

The Data Pointer registers, D0 and D1, provide the source addresses for all external data transfers.

The P Pointer register is used in Field Selection operations with the working registers.

3.3.4 Program Counter and Return Stack

The program counter points to the next instruction to be executed by the CPU. It can be accessed only using jump, gosub, and return instructions.

The current value of the program counter is automatically pushed onto the 8-level subroutine return stack when a gosub instruction is executed or an interrupt occurs. A 20-bit value is automatically popped off the return stack into the program counter when a return instruction is executed. The return stack can also be manipulated through use of the push (RSTK=C) and pop (C=RSTK) instructions.

3.3.5 Status Bits

Additional program control is provided by the 16-bit Program Status register and the 4-bit Hardware Status register. Each Status bit can be individually set, reset, and tested.

The operating system uses the upper 4 Program Status bits to indicate the state of the operating system. The remaining 12 Program Status bits are generally available to applications software, and may be manipulated collectively as the ST register.

The four Hardware Status bits are set (but not cleared) by hardware-related events, and must therefore be cleared beforehand in order to detect a particular occurrence. They are individually accessible by name. The Module Pulled bit (MP) is set when a module is pulled from or added to the machine. The Sticky Bit (SB) is set when a non-zero bit or digit shifts off the right end of a working register as the result of a shift right instruction, or the least significant nibble of a working register is non-zero prior to a shift right circular instruction. The Service Request (SR) bit is set as a result of a response to the SREQ? instruction (see section 2.5). The external Module Missing bit is set by execution of a "00" opcode (RTNSXM instruction). Since the BUS[0:3] is precharged low, the CPU will receive a RTNSXM instruction if no device responds to a PC READ bus command.

PROGRAM STATUS: 16 bits

Bits	Usage
15 thru 12	Indicate state of operating system
11 thru 0	Available to programs, may be manipulated as the ST register

HARDWARE STATUS: 4 bits

Bit	Symbol	Name
3	MP	Module Pulled
2	SR	Service Request
1	SB	Sticky Bit
0	XM	External Module Missing

3.3.6 Input/Output Registers

The Input and Output Registers provide the CPU with general purpose I/O. This consists of a 16-bit Input Register with interrupt capability (see section 3.5) and a 12-bit Output Register.

Data read from the Input Register corresponds to the logic levels sensed on the 16 input lines, IR[0:15]. The Input Register lines are passively held low so that if an input line is not driven a zero level will be sensed. Data written to the Output Register is driven onto the 12 output lines OR[0:11].

The Input and Output Registers are used to form the keyboard matrix. The Output Register lines OR[0:3] drive the 4 key-rows. The Input Register lines IR[0:13] sense the 14 key-columns. The most significant bit of the Input Register (IR15) is dedicated to the On-key and has additional interrupt capability (see section 4.4.2). When a key is pushed its key-row is shorted to its key-column, and the logic level driven on the Output Register line can be read from the Input Register bit. If the Output Register line is high and interrupts are enabled, pushing the key will result in an interrupt.

When one of the 8 lower order bits of the Output Register is low, the Output Register line will be actively driven low briefly each instruction cycle, then passively held low. This limits the current and provides a deterministic state when Output Register lines are driving different logic levels and are shorted. Output Register lines are shorted when 2 or more keys in the same column are depressed at the same time. This will result in a logic one level on the corresponding bit of the Input Register if one of the Output Register lines is high. The remaining 4 Output Register lines, OR[8:11] are actively driven at all times.

The 5 Output Register lines OR[0:4] also drive the DAISYIN lines of each I/O port. During configuration the operating system individually selects each I/O port by driving its DAISYIN line high (see section 8.3).

The 2 Output Register lines OR[10:11] drive the piezo-electric beeper. Two beeper loudness settings are obtained by using one of two Output Register lines.

3.3.7 Loading Data from Memory

When data is read from an external device into a register, the CPU places the lowest addressed nibble in the least significant nibble of the register. For example, if the data shown below in memory is read into the C register using the C=DAT1 4 instruction, the data in the register will be arranged as shown above.

```
 Memory
Location  Value              C   Register
--------  -----        +---------------------------+
  1000      0          |   |       | 3 | 2 | 1 | 0 |
  1001      1          +---------------------------+
  1002      2            15 . . .    3   2   1   0
  1003      3
    .
    .
    .
```

This principle also applies to loading constants into a CPU register such as C, D0,or D1, since the CPU must read the constant from the instruction opcode in memory. For example, the instruction LCHEX 3210 produces the opcode 330123 and the C register is loaded as shown above.

3.3.8 Storing Data in Memory

When data is written from a register to an external device, the CPU places the least significant nibble of the register in the first addressed nibble of the memory location. For example, if the data shown above in the C register is written to memory using the DAT1=C 4 instruction, the data will be written to memory as shown.

3.4 HEX/DEC Modes

All arithmetic operations, except for those listed below, are performed according to the HEX or DEC mode setting. The mode is set using the SETHEX or SETDEC instruction. The following operations are performed in HEX regardless of the mode setting.

C+P+1	
D0=D0+ n	D0=D0- n
D1=D1+ n	D1=D1- n
P=P+1	P=P-1

3.5 Interrupt System

The 1LF2 CPU can be interrupted in the following manners:

1. Pulling the *INT line low (module interrupt).
2. Shorting the least significant bit of the Output Register to the most significant bit of the Input Register (hardware "ON"-key).
3. An Input Register line (bits 0-15) being pulled high.

If interrupted, the CPU completes the current instruction, pushes the PC onto the return stack and sets the PC to the interrupt routine starting address of 0000F (where F is in the least significant nibble of the PC). Interrupts are disabled upon entry into the interrupt routine and are not enabled until a Return From Interrupt (RTI) instruction is executed. Each interrupt type operates in a somewhat different manner and is described separately.

3.5.1 *INT Interrupt

Pulling the *INT line low causes a non-maskable interrupt. When *INT is asserted the CPU enters the interrupt routine and disables all interrupts. When a RTI instruction is issued all interrupts may be processed immediately, including the *INT line even if it remains low from the previous interrupt.

3.5.2 Hardware "On-key" Interrupt

During the last cycle of every instruction the least significant bit of the Output Register (OR0) is driven to a one. This does not modify the contents of the Output Register, and the previous value is driven after this cycle. If the MSB of the Input Register (IR15) is driven high (On-key down) during this period, an interrupt is executed. This interrupt cannot be disabled outside of the interrupt routine.

Once a RTI instruction has been issued the CPU cannot be interrupted again by this Input Register line or any others until all Input Register lines are low. The CPU can be interrupted by the *INT line regardless of the state of the Input Register lines as long as a RTI instruction has been issued.

3.5.3 Input Register Interrupt

At any time except the time period allocated to the hardware "On-key", the CPU may be interrupted by any Input Register line pulled high. This interrupt may be disabled outside the interrupt routine by using the Interrupt Off (INTOFF) instruction, and re-enabled using the Interrupt On (INTON) instruction.

The CPU cannot be interrupted again by any Input Register line until a RTI instruction is issued and all Input Register lines are again low.

3.6 CPU Power-down; Wake-up

The CPU supports a low-power standby mode. To place the system in standby, a SHUTDN instruction is issued. The CPU then issues the SHUTDOWN bus command and on the ensuing cycle, stops *STR and its own clock. All CPU resident memory is preserved during standby. In standby mode the CPU holds the Bus lines in the conditions as follows:

BUS[0:3]	Low - through a high impedance
*STR	High - through a high impedance
*CD	High - through a high impedance

The CPU is brought out of standby mode when either an Input Register line goes high or the *CD line is driven low. On wake-up the CPU starts it's clock and immediately drives *CD low with a NOP bus command. If the CPU detects that a severe low voltage condition has occurred while it was in standby, the PC is set to zero and hex mode arithmetic is asserted. A LOAD PC bus command is issued after the NOP and the current CPU PC is loaded into all devices' local PCs. At this point the standard instruction fetch sequence in initiated. If the CPU was awakened by the Input Register and interrupts are enabled a normal interrupt will occur prior to the first instruction fetch. If a SHUTDN instruction is executed with the least significant bit of the Output Register (OR0) set at a "0" the CPU will wake-up immediately, set its PC to zero, assert hex mode arithmetic, and send out the PC. This avoids a situation where the CPU is shutdown and the keyboard cannot wake it up.

```
+---------------------------------------------+------------------+
|                                             |                  |
|   HP-71 ASSEMBLER INSTRUCTION SET           |   CHAPTER  4     |
|                                             |                  |
+---------------------------------------------+------------------+
```

This chapter describes the HP-71 assembler instruction set. The instruction mnemonics shown are those provided by the assembler used by the HP-71 software development team (which is available by special arrangement with HP). Almost all the mnemonics shown are also supported by the HP-71 FORTH/Assembler ROM.

4.1 Instruction Syntax

4.1.1 Labels and Symbols

A label is a symbolic name for a numeric value. A label acquires its value by appearing in the label field of certain statements. The word "symbol" is a general term for a label, and the two are used interchangeably.

Labels are one to six alphanumeric characters with the following restrictions: the characters comma (,), space ( ) and right parenthesis are prohibited and the first character cannot be equal sign (=), sharp (#), single quote ('), left parenthesis, or the digits 0 through 9.

A label may be immediately preceded by an equal sign which declares the label to be an external symbol. An external symbol defined in one module may be referenced as an external symbol by another module. Such references are resolved when the modules are linked together. Certain HP-71 assemblers, such as the FORTH/ASSEMBLER ROM, have no associated linker and therefore do not support external symbols. In this case, any leading equal sign is ignored.

When a label is used as part of an expression, parentheses are required to delineate it. That is, AD1-10 is a label but (AD1)-10 is a computed expression.

4.1.2 Comments

A comment line begins with an asterisk (*) in column one, and may occur anywhere. An in-line comment may begin with any non-blank character and must follow the modifier field of an instruction (or the opcode if no modifier is required).

4.1.3 Expressions

Wherever an expression may appear in the modifier field of an instruction, it is represented by the symbol "expr" in the instruction descriptions below. Expressions consist of:

EXPRESSION COMPONENTS

Item	Examples
decimal constants	23434
hexadecimal constants	#1FF0 (less than #100000)
ascii constants	\AB\ (3 or less characters) 'AB' (3 or less characters)
operators	+ addition - subtraction % *256+ * multiplication / integer division ^ exponentiate & and ! or
*	Current assembly program counter
label	Symbol defined in the label field of an instruction
(expression)	Parenthesized expression

Two classes of instructions require a modifier field which contains a constant of a specific type that does not conform to the above rules. These are:

a) String constant which can exceed 3 characters

```
LCASC     'ascii'       or
LCASC     \ascii\

NIBASC    'ascii'       or
NIBASC    \ascii\
```

b) Unconditional Hex constant

```
LCHEX     4FFFFF
NIBHEX    4FFFFF
```

4.1.4 Sample Line Image

The format below is the recommended column alignment; however, the assembler is "free format" and only a space is required to delimit the different fields. A label, if present, must start by column 2.

```
1      8       15            31                                  80
v      v       v             v                                    v
---------------------------------------------------------------------
label  opcode  modifier      comments
---------------------------------------------------------------------
```

4.2 Explanation of Symbols

In the following descriptions of the HP-71 assembler mnemonics, these symbols have the following meanings unless specified otherwise. In particular, note the symbols used to indicate the various values encoded within the assembled opcodes.

- a — The hex digit used to encode the field selection in the assembled opcode of an instruction. See the Field Select Table in the next section for details.
- b — The hex digit used to encode the field selection in the assembled opcode of an instruction. See the Field Select Table in the next section for details.
- d — The number of digits represented by a field selection field. Used in calculating the execution cycle time

of some instructions. See the Field Select Table in the next section for details. When used in an extended field selection fsd, represents an expression which indicates the number of nibbles of the register that will be affected by the instruction, proceeding from the low-order nibble to higher-order nibbles.

expr	An expression that evaluates to an absolute or relocatable value, usually less than or equal to 5 nibbles in length.
fs	Field selection symbol. See the Field Select Table in the next section for details.
fsd	Extended field selection symbol. Represents either a normal field selection symbol fs, or an expression that gives the number of digits d of the register that will be affected by the instruction, proceeding from the low-order nibble to higher-order nibbles.
hh	Two-digit hex constant, such as 08 or F2. Within an opcode represents the hex digits used to store the value of the expression in the opcode in reverse order (see "Loading Data From Memory").
hhhh	Four-digit hex constant, such as 38FE. Within an opcode, represents the hex digits used to store the value of the expression in the opcode in reverse order (see "Loading Data From Memory").
hhhhh	Five-digit hex constant, such as 308FE. Within an opcode, represents the hex digits used to store the value of the expression in the opcode in reverse order (see "Loading Data From Memory").
label	A symbol defined in the label field of an instruction.
m	A one-digit decimal integer constant.
n	Represents an expression that evaluates to a 1-nibble value, unless specified otherwise. Within an opcode, represents the hex digit used to store the assembled value of the expression in the opcode.
nn	Represents an expression that evaluates to a 2-nibble value, unless specified otherwise. Within an opcode, represents the hex digits used to store the assembled value of the expression in the opcode.
nnnn	Represents an expression that evaluates to a 4-nibble value, unless specified otherwise. Within an opcode,

represents the hex digits used to store the assembled value of the expression in the opcode.

nnnnn Represents an expression that evaluates to a 5-nibble value, unless specified otherwise. Within an opcode, represents the hex digits used to store the assembled value of the expression in the opcode.

4.2.1 Field Select Table

The following symbols are used in the instruction descriptions to denote the various possible field selections.

There are two ways in which field selection is encoded in the opcode of an instruction. These two patterns are shown in the table below, and are designated by the letter 'a' or 'b' in the opcode value given in the mnemonic descriptions below.

FIELD SELECT TABLE

Field	Name and Description	Opcode Representation (a)	Opcode Representation (b)	Number of Digits (d)
P	Pointer Field. Digit specified by P pointer register.	0	8	1
WP	Word-through-Pointer Field. Digits 0 through (P).	1	9	(P+1)
XS	Exponent Sign Field. Digit 2.	2	A	1
X	Exponent Field. Digits 0 - 2.	3	B	3
S	Sign Field. Digit 15.	4	C	1
M	Mantissa Field. Digits 3 - 14.	5	D	12
B	Byte Field. Digits 0 - 1.	6	E	2
W	Word Field. All digits.	7	F	16

The above field selects generally share the same opcode, with one nibble of the opcode containing the 'a' or 'b' value as specified in the table. The 'A' field select, however, generally is specified by a different opcode altogether. This has the effect of shortening and speeding up execution of 'A' field select manipulations, optimizing the computer for address and 5-nibble calculations.

4.3 Instruction Set Overview

The following pages briefly summarize the HP-71 instruction set. For further details please refer to the Mnemonic Dictionary which follows this summary.

4.3.1 GOTO Instructions

```
----
 l   = Statement Label
----

GOTO   label     Short unconditional branch
GOC    label     Short branch if Carry set
GONC   label     Short branch if no Carry set
GOLONG label     Long GOTO
GOVLNG label     Very long GOTO
GOYES  label     Short branch if test true (must
                   follow a Test Instruction)
```

4.3.2 GOSUB Instructions

```
GOSUB  label     Short transfer to subroutine
GOSUBL label     Long GOSUB
GOSBVL label     Very long GOSUB
```

4.3.3 Subroutine Returns

```
RTN              Unconditional return
RTNSC            Return and set Carry
RTNCC            Return and clear Carry
RTNSXM           Return and set XM bit (Module Missing)
RTI              Return and enable interrupts
RTNC             Return if Carry set
```

```
RTNNC             Return if no Carry set
RTNYES            Return if test true (must follow a
                    Test Instruction)
```

4.3.4 Test Instructions

All test instructions must be followed with a GOYES or a RTNYES instruction. Although they appear to be two statements, in fact they combine to be one. Each test adjusts the Carry bit when performed: Carry is set if the test is true, and cleared if false.

4.3.4.1 Register Tests

```
----
 r,s = A,B,C  or  (r,s) = (C,D),(D,C)
 fs  = Field Select
----

?r=s   fs         Equal
?r#s   fs         Not equal
?r=0   fs         Equal to zero
?r#0   fs         Not equal to zero
?r>s   fs         Greater than
?r<s   fs         Less than
?r>=s  fs         Greater than or equal
?r<=s  fs         Less than or equal
```

4.3.4.2 P Pointer Tests

```
----
 0 <= n <= 15
----

?P=    n          Is P Pointer equal to n?
?P#    n          P Pointer not equal to n?
```

4.3.4.3 Hardware Status Bit Tests

```
?XM=0             Module Missing bit equal to zero?
?SB=0             Sticky Bit equal to zero?
?SR=0             Service Request bit equal to zero?
?MP=0             Module Pulled bit equal to zero?
```

4.3.4.4 Program Status Bit Tests

```
----
 0 <= n <= 15
----
```

```
?ST=1  n        Status n equal to 1?
?ST=0  n        Status n equal to 0?
?ST#1  n        Status not equal to 1?
?ST#0  n        Status not equal to 0?
```

4.3.5 P Pointer Instructions

```
----
 0 <= n <= 15
----

P=     n        Set P Pointer to n
P=P+1           Increment P Pointer, adjust Carry
P=P-1           Decrement P Pointer, adjust Carry
C+P+1           Add P Pointer plus one to A-field of C
CPEX   n        Exchange P Pointer with nibble n of C
P=C    n        Copy nibble n of C into P Pointer
C=P    n        Copy P Pointer into nibble n of C
```

4.3.6 Status Instructions

4.3.6.1 Program Status

```
----
 0 <= n <= 15
----

ST=1   n        Set Status n to 1
ST=0   n        Set Status n to 0
CSTEX           Exchange X field of C with Status 0-11
C=ST            Copy Status 0-11 into X field of C
ST=C            Copy X field of C into Status 0-11
CLRST           Clear Status 0-11
```

4.3.6.2 Hardware Status

```
SB=0            Clear Sticky Bit
SR=0            Clear Service Request bit (see SREQ?)
MP=0            Clear Module-Pulled bit
XM=0            Clear External Module Missing bit
CLRHST          Clear all 4 Hardware Status bits
```

4.3.7 System Control

SETHEX	Set arithmetic mode to hexadecimal
SETDEC	Set arithmetic mode to decimal
SREQ?	Sets Service Request bit if service has has been requested. C(0) shows what bit(s) are pulled high (if any)
C=RSTK	Pop return stack into A-field of C
RSTK=C	Push A-field of C onto return stack
CONFIG	Configure
UNCNFG	Unconfigure
RESET	Send Reset command to system bus
BUSCC	Send Bus command C onto system bus
SHUTDN	Stop CPU here (sleeps until wake-up)
C=ID	Request chip ID into A-field of C
INTOFF	Disable interrupts (doesn't affect ON-key or module-pulled interrupts)
INTON	Enable interrupts

4.3.8 Keyscan Instructions

OUT=C	Copy X field of C to OUTput register
OUT=CS	Copy nibble 0 of C to OUTput register
A=IN	Copy INput register to lower 4 nibbles of A
C=IN	Copy INput register to lower 4 nibbles of C

4.3.9 Register Swaps

----
s = R0,R1,R2,R3,R4

AsEX	Exchange register A with s
CsEX	Exchange register C with s
A=s	Copy s to register A
C=s	Copy s to register C
s=A	Copy register A to s
s=C	Copy register C to s

4.3.10 Data Pointer Manipulation

----
d = D0,D1

```
 1 <= n <= 16
 expr <= 5 nibbles
----

AdEX              Exchange Data ptr d with A-field of A
CdEX              Exchange Data ptr d with A-field of C
AdXS              Exchange lower 4 nibs of Data ptr d with
                    lower 4 nibs of A
CdXS              Exchange lower 4 nibs of Data ptr d with
                    lower 4 nibs of C
d=A               Copy A-field of A to Data pointer d
d=C               Copy A-field of C to Data pointer d
d=AS              Copy lower 4 nibs of A to lower 4 nibs
                    of Data pointer d
d=CS              Copy lower 4 nibs of C to lower 4 nibs
                    of Data pointer d
d=d+   n          Increment Data pointer d by n
d=d-   n          Decrement Data pointer d by n
d=HEX  hh         Load hh   into lower 2 nibs of Data ptr d
d=HEX  hhhh       Load hhhh into lower 4 nibs of Data ptr d
d=HEX  hhhhh      Load hhhhh into Data ptr d
d=(2)  nn         Load nn into lower 2 nibs of Data ptr d
                    (any overflow is ignored)
d=(4)  nnnn       Load nnnn into lower 4 nibs of Data ptr d
                    (any overflow is ignored)
d=(5)  nnnnn      Load nnnnn into Data ptr d  (any overflow
                    is ignored)
```

4.3.11 Data Transfer

```
----
 fsd = Field select fs, or d (# of digits).  If d, then
       the copy starts at nibble 0 of the working register.
 1 <= d <= 16
----

A=DAT0 fsd        Copy data from memory addressed by D0 into
                    A, field selected
C=DAT0 fsd        Copy data from memory addressed by D0 into
                    C, field selected
A=DAT1 fsd        Copy data from memory addressed by D1 into
                    A, field selected
C=DAT1 fsd        Copy data from memory addressed by D1 into
                    C, field selected
DAT0=A fsd        Copy data from A into memory addressed by
                    D0, field selected
DAT0=C fsd        Copy data from C into memory addressed by
                    D0, field selected
DAT1=A fsd        Copy data from A into memory addressed by
```

```
                        D1, field selected
DAT1=C fsd            Copy data from C into memory addressed by
                        D1, field selected
```

4.3.12 Load Constants

```
LCHEX  hhhhhhhh  Load hex constant into C (1 to 16 digits)
LC(m)  expr      Load the m-nibble constant into C
LCASC  'ascii'   Load up to 8 ASCII characters into C
LCASC  \ascii\   Load up to 8 ASCII characters into C
```

4.3.13 Shift Instructions

Note that right shifts (circular or non-circular) will set the Sticky Bit (SB=1) if a nonzero bit is shifted off to the right. Otherwise, SB is unchanged.

```
----
 r = A,B,C,D
 fs = Field Select
----

rSL    fs       Shift register r fs field Left  1 nibble
rSR    fs       Shift register r fs field Right 1 nibble
rSLC            Shift register r Left  Circular 1 nibble
rSRC            Shift register r Right Circular 1 nibble
rSRB            Shift register r Right          1 bit
```

4.3.14 Logical Operations

Logical operations are bit-wise.

```
----
 r,s = A,B,C  or  (r,s) = (C,D),(D,C)
 fs = Field Select
----

r=r&s  fs       r AND s into r, field selected
r=r!s  fs       r OR  s into r, field selected
```

4.3.15 Arithmetics

The two groups of arithmetics differ in the range of registers available. In the first group (General usage) almost all combinations of the four working registers are possible; however, in the second group (Restricted usage) only a few select combinations are possible.

4.3.15.1 General Usage

```
----
 r,s = A,B,C  or (r,s) = (C,D),(D,C)
 fs  = Field Select
----

r=0     fs        Set r to zero
r=r+r   fs        Double r, adjust Carry
r=r+1   fs        Increment r by 1, adjust Carry
r=r-1   fs        Decrement r by 1, adjust Carry
r=-r    fs        10'S complement or 2'S complement, Carry
                    set if r#0, else clear
r=-r-1 fs         9'S complement or 1'S complement
                    Carry always cleared
r=r+s   fs        Sum r and s into r, adjust Carry
s=r+s   fs        Sum r and s into s, adjust Carry
r=s     fs        Copy s into r
s=r     fs        Copy r into s
rsEX    fs        Exchange r and s
```

4.3.15.2 Restricted Usage

```
----
 (r,s) = (A,B),(B,C),(C,A),(D,C)
----

r=r-s   fs        Difference of r and s into r, adjust Carry
r=s-r   fs        Difference of s and r into r, adjust Carry
s=s-r   fs        Difference of s and r into s, adjust Carry
```

4.3.16 No-Op Instructions

Execution of a No-Op affects no CPU registers except for the PC.

```
NOP3              Three nibble No-Op
NOP4              Four nibble No-Op
NOP5              Five nibble No-Op
```

4.3.17 Pseudo-Ops

4.3.17.1 Data Storage Allocation

```
----
 1 <= n <= 8
----

BSS     nnnnn     Allocate nnnnn number of zero nibs

CON(m) expr       Generate m-nibble constant (digits are
                    reversed in the opcode)
REL(m) expr       Generate m-nibble relative constant (digits
                    reversed in the opcode)

NIBASC 'ascii'    Generate ascii characters, byte reversed
NIBASC \ascii\    Generate ascii characters, byte reversed
NIBHEX hhhh       Generate hexadecimal digits hhhh  (digits
                    are not reversed in the opcode)
```

4.3.17.2 Conditional Assembly

```
name  IF     expr       Start conditional assembly until ELSE or
                          ENDIF if flag expr was set on invocation
                          of assembler (optional use of name allows
                          nesting of IF's)
name  ELSE              Conditional assembly if IF test was false
name  ENDIF             Ends conditional assembly started by IF
```

4.3.17.3 Listing Formatting

```
EJECT             Force new page in the assembly listing
STITLE text       Force new page, set subtitle value to text
TITLE  text       Set title value to text
```

4.3.17.4 Symbol Definition

```
label EQU   nnnnn     Defines label to have the value expr
```

4.3.17.5 Assembly Mode

```
ABS    nnnnn      Specify absolute assembly at adress given
END               Marks end of the assembly source
```

4.4 Mnemonic Dictionary

This section contains a description of each HP-71 assembler instruction or pseudo-op. The description shows the binary opcode generated by the mnemonic, if any, as well as the execution cycle time required if the mnemonic is an executable instruction.

The symbols used in these descriptions are explained in the "Explanation of Symbols" section earlier in this chapter.

```
******************************************************
*                      MNEMONICS                     *
******************************************************
```

```
?A#0   fs  -  Test for A not equal to 0
---------
fs = A                        opcode:  8ACyy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)      opcode:  9aCyy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)
```

Test whether the fs field of A is not equal to 0. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

```
?A#B   fs  -  Test for A not equal to B
---------
fs = A                        opcode:  8A4yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)      opcode:  9a4yy
                              cycles:   13 + d (GO/RTNYES)
```

6 + d (NO)

Test whether the fs field of A is not equal to the fs field of B. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?A#C fs - Test for A not equal to C

```
fs = A                      opcode:  8A6yy
                            cycles:   13 + d (GO/RTNYES)
                                       6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)    opcode:  9a6yy
                            cycles:   13 + d (GO/RTNYES)
                                       6 + d (NO)
```

Test whether the fs field of A is not equal to the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?A<=B fs - Test for A less than or equal to B

```
fs = A                      opcode:  8BCyy
                            cycles:   13 + d (GO/RTNYES)
                                       6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)    opcode:  9bCyy
                            cycles:   13 + d (GO/RTNYES)
                                       6 + d (NO)
```

Test whether the fs field of A is less than or equal to the fs field of B. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

```
?A<B   fs   -  Test for A less than B
---------
fs = A                        opcode:  8B4yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)      opcode:  9b4yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)
```

Test whether the fs field of A is less than the fs field of B. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

```
?A=0   fs   -  Test for A equal to 0
---------
fs = A                        opcode:  8A8yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)      opcode:  9a8yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)
```

Test whether the fs field of A is equal to 0. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

```
?A=B   fs   -  Test for A equal to B
---------
fs = A                        opcode:  8A0yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)      opcode:  9a0yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)
```

Test whether the fs field of A is equal to the fs field of B. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?A=C fs - Test for A equal to C
----

fs = A opcode: 8A2yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W) opcode: 9a2yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

Test whether the fs field of A is equal to the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?A>=B fs - Test for A greater than or equal to B
----

fs = A opcode: 8B8yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W) opcode: 9b8yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

Test whether the fs field of A is greater than or equal to the fs field of B. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?A>B fs - Test for A greater than B
----

fs = A opcode: 8B0yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W) opcode: 9b0yy
cycles: 13 + d (GO/RTNYES)

6 + d (NO)

Test whether the fs field of A is greater than the fs field of B. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?B#0 fs - Test for B not equal to 0
----

fs = A	opcode: 8ADyy cycles: 13 + d (GO/RTNYES) 6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode: 9aDyy cycles: 13 + d (GO/RTNYES) 6 + d (NO)

Test whether the fs field of B is not equal to 0. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?B#A fs - Test for B not equal to A
----

fs = A	opcode: 8A4yy cycles: 13 + d (GO/RTNYES) 6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode: 9a4yy cycles: 13 + d (GO/RTNYES) 6 + d (NO)

Test whether the fs field of B is not equal to the fs field of A. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

```
?B#C  fs  -  Test for B not equal to C
---------
fs = A                        opcode:  8A5yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)      opcode:  9a5yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)
```

Test whether the fs field of B is not equal to the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

```
?B<=C  fs  -  Test for B less than or equal to C
---------
fs = A                        opcode:  8BDyy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)      opcode:  9bDyy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)
```

Test whether the fs field of B is less than or equal to the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

```
?B<C  fs  -  Test for B less than C
---------
fs = A                        opcode:  8B5yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)      opcode:  9b5yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)
```

Test whether the fs field of B is less than the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

```
?B=0   fs   -  Test for B equal to 0
---------
fs = A                        opcode:  8A9yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)      opcode:  9a9yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)
```

Test whether the fs field of B is equal to 0. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

```
?B=A   fs   -  Test for B equal to A
---------
fs = A                        opcode:  8A0yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)      opcode:  9a0yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)
```

Test whether the fs field of B is equal to the fs field of A. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

```
?B=C   fs   -  Test for B equal to C
---------
fs = A                        opcode:  8A1yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)      opcode:  9a1yy
                              cycles:   13 + d (GO/RTNYES)
```

6 + d (NO)

Test whether the fs field of B is equal to the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?B>=C fs - Test for B greater than or equal to C

```
fs = A                        opcode:  8B9yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)      opcode:  9b9yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)
```

Test whether the fs field of B is greater than or equal to the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?B>C fs - Test for B greater than C

```
fs = A                        opcode:  8B1yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)      opcode:  9b1yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)
```

Test whether the fs field of B is greater than the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

```
?C#0   fs  -  Test for C not equal to 0
---------
fs = A                        opcode:  8AEyy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)      opcode:  9aEyy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)
```

Test whether the fs field of C is not equal to 0. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

```
?C#A   fs  -  Test for C not equal to A
---------
fs = A                        opcode:  8A6yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)      opcode:  9a6yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)
```

Test whether the fs field of C is not equal to the fs field of A. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

```
?C#B   fs  -  Test for C not equal to B
---------
fs = A                        opcode:  8A5yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)      opcode:  9a5yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)
```

Test whether the fs field of C is not equal to the fs field of B. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

```
?C#D   fs  -  Test for C not equal to D
---------
fs = A                         opcode:  8A7yy
                               cycles:   13 + d (GO/RTNYES)
                                          6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)       opcode:  9a7yy
                               cycles:   13 + d (GO/RTNYES)
                                          6 + d (NO)
```

Test whether the fs field of C is not equal to the fs field of D. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

```
?C<=A  fs  -  Test for C less than or equal to A
---------
fs = A                         opcode:  8BEyy
                               cycles:   13 + d (GO/RTNYES)
                                          6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)       opcode:  9bEyy
                               cycles:   13 + d (GO/RTNYES)
                                          6 + d (NO)
```

Test whether the fs field of C is less than or equal to the fs field of A. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

```
?C<A   fs  -  Test for C less than A
---------
fs = A                         opcode:  8B6yy
                               cycles:   13 + d (GO/RTNYES)
                                          6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)       opcode:  9b6yy
                               cycles:   13 + d (GO/RTNYES)
```

6 + d (NO)

Test whether the fs field of C is less than the fs field of A. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?C=0 fs - Test for C equal to 0
----

fs = A	opcode:	8AAyy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9aAyy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of C is equal to 0. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?C=A fs - Test for C equal to A
----

fs = A	opcode:	8A2yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)
fs = (P,WP,XS,X,S,M,B,W)	opcode:	9a2yy
	cycles:	13 + d (GO/RTNYES)
		6 + d (NO)

Test whether the fs field of C is equal to the fs field of A. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

```
?C=B   fs  -  Test for C equal to B
---------
fs = A                          opcode:  8A1yy
                                cycles:   13 + d (GO/RTNYES)
                                           6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)        opcode:  9a1yy
                                cycles:   13 + d (GO/RTNYES)
                                           6 + d (NO)
```

Test whether the fs field of C is equal to the fs field of B. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

```
?C=D   fs  -  Test for C equal to D
---------
fs = A                          opcode:  8A3yy
                                cycles:   13 + d (GO/RTNYES)
                                           6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)        opcode:  9a3yy
                                cycles:   13 + d (GO/RTNYES)
                                           6 + d (NO)
```

Test whether the fs field of C is equal to the fs field of D. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

```
?C>=A  fs  -  Test for C greater than or equal to A
---------
fs = A                          opcode:  8BAyy
                                cycles:   13 + d (GO/RTNYES)
                                           6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)        opcode:  9bAyy
                                cycles:   13 + d (GO/RTNYES)
                                           6 + d (NO)
```

Test whether the fs field of C is greater than or equal to the fs field of A. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?C>A fs - Test for C greater than A
----

fs = A opcode: 8B2yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W) opcode: 9b2yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

Test whether the fs field of C is greater than the fs field of A. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?D#0 fs - Test for D not equal to 0
----

fs = A opcode: 8AFyy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W) opcode: 9aFyy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

Test whether the fs field of D is not equal to 0. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?D#C fs - Test for D not equal to C
----

fs = A opcode: 8A7yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W) opcode: 9a7yy
cycles: 13 + d (GO/RTNYES)

6 + d (NO)

Test whether the fs field of D is not equal to the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?D<=C fs - Test for D less than or equal to C

```
fs = A                        opcode:  8BFyy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)      opcode:  9bFyy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)
```

Test whether the fs field of D is less than or equal to the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?D<C fs - Test for D less than to C

```
fs = A                        opcode:  8B7yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)      opcode:  9b7yy
                              cycles:   13 + d (GO/RTNYES)
                                         6 + d (NO)
```

Test whether the fs field of D is less than the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?D=0   fs - Test for D equal to 0

```
fs = A                      opcode:  8AByy
                            cycles:   13 + d (GO/RTNYES)
                                       6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)    opcode:  9aByy
                            cycles:   13 + d (GO/RTNYES)
                                       6 + d (NO)
```

Test whether the fs field of D is equal to 0. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?D=C   fs - Test for D equal to C

```
fs = A                      opcode:  8A3yy
                            cycles:   13 + d (GO/RTNYES)
                                       6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)    opcode:  9a3yy
                            cycles:   13 + d (GO/RTNYES)
                                       6 + d (NO)
```

Test whether the fs field of D is equal to the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?D>=C fs - Test for D greater than or equal to C

```
fs = A                      opcode:  8BByy
                            cycles:   13 + d (GO/RTNYES)
                                       6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W)    opcode:  9bByy
                            cycles:   13 + d (GO/RTNYES)
                                       6 + d (NO)
```

Test whether the fs field of D is greater than or equal to the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?D>C fs - Test for D greater than C
----

fs = A — opcode: 8B3yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

fs = (P,WP,XS,X,S,M,B,W) — opcode: 9b3yy
cycles: 13 + d (GO/RTNYES)
6 + d (NO)

Test whether the fs field of D is greater than the fs field of C. Must be followed by a GOYES or RTNYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?MP=0 - Test Module Pulled bit (MP)
----

opcode: 838yy
cycles: 13 (GO/RTNYES)
6 (NO)

Test whether the Module Pulled bit (MP) is zero. This hardware status bit is set whenever a module-pulled interrupt occurs (the *INT line of the CPU is pulled high), and must be explictly cleared by the MP=0 mnemonic. See the "HP-71 Hardware Specification" for more information. Must be followed by a RTNYES or GOYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?P# n - Test if P pointer not equal to n
----

opcode: 88nyy
cycles: 13 (GO/RTNYES)
6 (NO)

Test whether the P pointer is not equal to n. Must be followed by a RTNYES or GOYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?P= n - Test if P pointer is equal to n
----

opcode: 89nyy
cycles: 13 (GO/RTNYES)
6 (NO)

Test whether the P pointer is equal to n. Must be followed by a RTNYES or GOYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?SB=0 - Test Sticky Bit (SB)
----

opcode: 832yy
cycles: 13 (GO/RTNYES)
6 (NO)

Test whether the Sticky Bit (SB) is zero. This hardware status bit is set on right shifts (circular or non-circular) when a non-zero nibble or bit is shifted off the end of the field. The Sticky Bit must be cleared explicitly by the SB=0 mnemonic. Must be followed by a RTNYES or GOYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?SR=0 - Test Service Request bit (SR) for zero
----

opcode: 834yy
cycles: 13 (GO/RTNYES)
6 (NO)

Test whether the Service Request bit (SR) is zero. This hardware status bit is set by the SREQ? mnemonic, and must be cleared explicitly by the SR=0 mnemonic. Must be followed by a RTNYES or GOYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?ST#0 n - Test status bit n not equal to 0
----

opcode: 87nyy
cycles: 14 (GO/RTNYES)
7 (NO)

Test whether Program Status bit n is set. Must be followed by a RTNYES or GOYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?ST#1 n - Test status bit n not equal to 1
----

opcode: 86nyy
cycles: 14 (GO/RTNYES)
7 (NO)

Test whether Program Status bit n is clear. Must be followed by a RTNYES or GOYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

?ST=0 n - Test status bit n equal to 0
----

opcode: 86nyy
cycles: 14 (GO/RTNYES)
7 (NO)

Test whether Program Status bit n is clear. Must be followed by a RTNYES or GOYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

```
?ST=1  n    - Test status bit n equal to 1
---------
                              opcode:  87nyy
                              cycles:   14 (GO/RTNYES)
                                         7 (NO)
```

Test whether Program Status bit n is set. Must be followed by a RTNYES or GOYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

```
?XM=0     - Test External Module Missing bit (XM)
---------
                              opcode:  831yy
                              cycles:   13 (GO/RTNYES)
                                         6 (NO)
```

Test the whether the External Module Missing bit (XM) is zero. This hardware status bit is set by the RTNSXM mnemonic, and must be explicitly cleared by the XM=0 mnemonic. Must be followed by a RTNYES or GOYES mnemonic. yy is determined by the following RTNYES or GOYES. Adjusts Carry.

```
A=-A   fs   -  Two's complement of A into A
---------
fs = A                        opcode:  F8
                              cycles:    7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  Bb8
                              cycles:    3 + d
```

Complement the specified fs field of A. Complement is two's complement if in HEX mode, ten's complement if in DEC mode. Carry is set if the field is not zero, else Carry is cleared.

A=-A-1 fs - One's complement of A into A

```
fs = A                        opcode:  FC
                              cycles:    7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  BbC
                              cycles:    3 + d
```

Perform a one's complement on the specified fs field of A. Carry is always cleared.

A=0 fs - Set A equal to 0

```
fs = A                        opcode:  D0
                              cycles:    7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  Ab0
                              cycles:    3 + d
```

Set the specified fs field of A to zero. Carry is not affected.

A=A!B fs - A OR B into A

```
fs = A                        opcode:  0EF8
                              cycles:    4 + d

fs = (P,WP,XS,X,S,M,B,W)      opcode:  0Ea8
                              cycles:    4 + d
```

Set the fs field of register A to its logical OR with the corresponding field of register B. Carry is not affected.

A=A!C fs - A OR C into A

```
fs = A                        opcode:  0EFE
                              cycles:    4 + d

fs = (P,WP,XS,X,S,M,B,W)      opcode:  0EaE
                              cycles:    4 + d
```

Set the fs field of register A to its logical OR with the corresponding field of register C. Carry is not affected.

A=A&B fs - A AND B into A

```
fs = A                        opcode:  0EF0
                              cycles:    4 + d

fs = (P,WP,XS,X,S,M,B,W)      opcode:  0Ea0
                              cycles:    4 + d
```

Set the fs field of register A to its logical AND with the corresponding field of register B. Carry is not affected.

A=A&C fs - A AND C into A

```
fs = A                        opcode:  0EF6
                              cycles:    4 + d

fs = (P,WP,XS,X,S,M,B,W)      opcode:  0Ea6
                              cycles:    4 + d
```

Set the fs field of register A to its logical AND with the corresponding field of register C. Carry is not affected.

A=A+1 fs - Increment A
----

fs = A	opcode:	E4
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Ba4
	cycles:	3 + d

Increment the specified fs field of register A by one. Adjusts Carry.

A=A+A fs - Sum of A and A into A
----

fs = A	opcode:	C4
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Aa4
	cycles:	3 + d

Double the specified fs field of register A. Adjusts Carry.

A=A+B fs - Sum of A and B into A
----

fs = A	opcode:	C0
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Aa0
	cycles:	3 + d

Set the specified fs field of register A to the sum of itself and the corresponding field of register B. Adjusts Carry.

```
A=A+C  fs  -  Sum of A and C into A
---------
fs = A                        opcode:  CA
                              cycles:    7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  AaA
                              cycles:    3 + d
```

Set the specified fs field of register A to the sum of itself and the corresponding field of register C. Adjusts Carry.

```
A=A-1  fs  -  Decrement A
---------
fs = A                        opcode:  CC
                              cycles:    7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  AaC
                              cycles:    3 + d
```

Decrement the specified fs field of register A by one. Adjusts Carry.

```
A=A-B  fs  -  A minus B into A
---------
fs = A                        opcode:  E0
                              cycles:    7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  Ba0
                              cycles:    3 + d
```

Set the specified fs field of register A to the difference between itself and the corresponding field of register B. Adjusts Carry.

A=A-C fs - A minus C into A
----

fs = A	opcode:	EA
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	BaA
	cycles:	3 + d

Set the specified fs field of register A to the difference between itself and the corresponding field of register C. Adjusts Carry.

A=B fs - Copy B to A
----

fs = A	opcode:	D4
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Ab4
	cycles:	3 + d

Copy the fs field of register B into the corresponding field of register A. Carry is not affected.

A=B-A fs - B minus A into A
----

fs = A	opcode:	EC
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	BaC
	cycles:	3 + d

Set the specified fs field of register A to the inverse difference between itself and the corresponding field of register B. Adjusts Carry.

A=C fs - Copy C to A

----

fs = A	opcode:	DA
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	AbA
	cycles:	3 + d

Copy the fs field of register C into the corresponding field of register A. Carry is not affected.

A=DAT0 fsd - Load A from memory

----

fs = A	opcode:	142
	cycles:	18
fs = B	opcode:	14A
	cycles:	15
fs = (P,WP,XS,X,S,M,W)	opcode:	152a
	cycles:	17 + d
fs = d	opcode:	15Ax (x=d-1)
	cycles:	16 + d

The amount of data (d nibbles) specified by fsd will be transferred from the memory address pointed to by D0 into the specified field of register A. The lowest-addressed nibble will be transferred into the lowest-order nibble of the register field, proceeding toward the higher-order nibbles. If fs = d, d nibbles are transferred into the register starting at nibble 0. See the section on "Loading Data From Memory" earlier in this chapter.

A=DAT1 fsd - Load A from memory

----

fs = A	opcode:	143
	cycles:	18
fs = B	opcode:	14B
	cycles:	15

```
fs = (P,WP,XS,X,S,M,W)        opcode:  153a
                              cycles:   17 + d

fs = d                        opcode:  15Bx (x=d-1)
                              cycles:   16 + d
```

The amount of data (d nibbles) specified by fsd will be transferred from the memory address pointed to by D1 into the specified field of register A. The lowest-addressed nibble will be transferred into the lowest-order nibble of the register field, proceeding toward the higher-order nibbles. If fs = d, d nibbles are transferred into the register starting at nibble 0. See the section on "Loading Data From Memory" earlier in this chapter.

A=IN - Load A with IN

```
                              opcode:  802
                              cycles:    7
```

Load the low-order 4 nibbles of the A register with the contents of the Input register.

A=R0 - Copy R0 to A

```
                              opcode:  110
                              cycles:   19
```

The contents of the scratch register R0 is copied to the working register A.

A=R1 - Copy R1 to A
----

opcode: 111
cycles: 19

The contents of the scratch register R1 is copied to the working register A.

A=R2 - Copy R2 to A
----

opcode: 112
cycles: 19

The contents of the scratch register R2 is copied to the working register A.

A=R3 - Copy R3 to A
----

opcode: 113
cycles: 19

The contents of the scratch register R3 is copied to the working register A.

A=R4 - Copy R4 to A
----

opcode: 114
cycles: 19

The contents of the scratch register R4 is copied to the working register A.

ABEX fs - Exchange Registers A and B
----

fs = A opcode: DC
cycles: 7

fs = (P,WP,XS,X,S,M,B,W) opcode: AbC
cycles: 3 + d

Exchange the fs fields of registers of A and B. Carry is not affected.

ACEX fs - Exchange Registers A and C
----

fs = A opcode: DE
cycles: 7

fs = (P,WP,XS,X,S,M,B,W) opcode: AbE
cycles: 3 + d

Exchange the fs fields of registers of A and C. Carry is not affected.

AD0EX - Exchange A and D0 (nibs 0-4)
----

opcode: 132
cycles: 8

Exchange the A field of register A with Data pointer D0. Carry is not affected.

AD0XS - Exchange A and D0 short (nibs 0-3)
----

opcode: 13A
cycles: 7

Exchange the lower 4 nibbles of A with the lower 4 nibbles of Data pointer D0. Carry is not affected.

AD1EX - Exchange A and D1 (nibs 0-4)
----

opcode: 133
cycles: 8

Exchange the A field of register A with Data pointer D1. Carry is not affected.

AD1XS - Exchange A and D1 short (nibs 0-3)
----

opcode: 13B
cycles: 7

Exchange the lower 4 nibbles of A with the lower 4 nibbles of Data pointer D1. Carry is not affected.

AR0EX - Exchange A and R0
----

opcode: 120
cycles: 19

Exchange the contents of the working register A and the scratch register R0.

AR1EX - Exchange A and R1
----

opcode: 121
cycles: 19

Exchange the contents of the working register A and the scratch register R1.

AR2EX - Exchange A and R2
----

opcode: 122
cycles: 19

Exchange the contents of the working register A and the scratch register R2.

AR3EX - Exchange A and R3
----

opcode: 123
cycles: 19

Exchange the contents of the working register A and the scratch register R3.

AR4EX - Exchange A and R4
----

opcode: 124
cycles: 19

Exchange the contents of the working register A and the scratch register R4.

```
ASL     fs   -   A Shift Left
---------
fs = A                          opcode:  F0
                                cycles:    7

fs = (P,WP,XS,X,S,M,B,W)        opcode:  Bb0
                                cycles:    3 + d
```

Shift the contents of the specified fs field of register A left one nibble, without affecting the rest of the register. The nibble shifted off the left end of the field is lost. The new low-order nibble of the field is zero. The Sticky Bit (SB) is not affected.

```
ASLC       - A Shift Left Circular
---------
                                opcode:  810
                                cycles:   21
```

Circular shift register A left one nibble. Operates on all 16 digits. The Sticky Bit (SB) is not affected.

```
ASR     fs   -   A Shift Right
---------
fs = A                          opcode:  F4
                                cycles:    7

fs = (P,WP,XS,X,S,M,B,W)        opcode:  Bb4
                                cycles:    3 + d
```

Shift the contents of the specified fs field of register A right one nibble, without affecting the rest of the register. The nibble shifted off the right end of the field is lost, but the Sticky Bit (SB) is set if the nibble was non-zero. The new high-order nibble of the field is zero.

ASRB - A Shift Right Bit
----

opcode: 81C
cycles: 20

Shift register A right one bit. Operates on all 16 digits. The bit shifted off the end is lost, but the Sticky Bit (SB) is set if it was non-zero. The new high-order bit of the register is zero.

ASRC - A Shift Right Circular
----

opcode: 814
cycles: 21

Circular shift register A right one nibble. Operates on all 16 digits. The Sticky Bit (SB) is set if the nibble shifted from low-order around to high-order position was non-zero.

B=-B fs - Two's complement of B into B
----

fs = A
opcode: F9
cycles: 7

fs = (P,WP,XS,X,S,M,B,W)
opcode: Bb9
cycles: 3 + d

Complement the specified fs field of B. Complement is two's complement if in HEX mode, ten's complement if in DEC mode. Carry is set if the field is not zero, else Carry is cleared.

```
B=-B-1 fs  -  One's complement of B into B
---------
fs = A                        opcode:  FD
                              cycles:    7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  BbD
                              cycles:    3 + d
```

Perform a one's complement on the specified fs field of B. Carry is always cleared.

```
B=0    fs  -  Set B equal to 0
---------
fs = A                        opcode:  D1
                              cycles:    7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  Ab1
                              cycles:    3 + d
```

Set the specified fs field of B to zero. Carry is not affected.

```
B=A    fs  -  Copy A to B
---------
fs = A                        opcode:  D8
                              cycles:    7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  Ab8
                              cycles:    3 + d
```

Copy the fs field of register A into the corresponding field of register B. Carry is not affected.

B=B!A fs - B OR A into B

```
fs = A                        opcode:  0EFC
                              cycles:    4 + d

fs = (P,WP,XS,X,S,M,B,W)      opcode:  0EaC
                              cycles:    4 + d
```

Set the fs field of register B to its logical OR with the corresponding field of register A. Carry is not affected.

B=B!C fs - B OR C into B

```
fs = A                        opcode:  0EF9
                              cycles:    4 + d

fs = (P,WP,XS,X,S,M,B,W)      opcode:  0Ea9
                              cycles:    4 + d
```

Set the fs field of register B to its logical OR with the corresponding field of register C. Carry is not affected.

B=B&A fs - B AND A into B

```
fs = A                        opcode:  0EF4
                              cycles:    4 + d

fs = (P,WP,XS,X,S,M,B,W)      opcode:  0Ea4
                              cycles:    4 + d
```

Set the fs field of register B to its logical AND with the corresponding field of register A. Carry is not affected.

B=B&C fs - B AND C into B

```
fs = A                          opcode:  0EF1
                                cycles:    4 + d

fs = (P,WP,XS,X,S,M,B,W)        opcode:  0Ea1
                                cycles:    4 + d
```

Set the fs field of register B to its logical AND with the corresponding field of register C. Carry is not affected.

B=B+1 fs - Increment B

```
fs = A                          opcode:  E5
                                cycles:    7

fs = (P,WP,XS,X,S,M,B,W)        opcode:  Ba5
                                cycles:    3 + d
```

Increment the specified fs field of register B by one. Adjusts Carry.

B=B+A fs - Sum of B and A into B

```
fs = A                          opcode:  C8
                                cycles:    7

fs = (P,WP,XS,X,S,M,B,W)        opcode:  Aa8
                                cycles:    3 + d
```

Set the specified fs field of register B to the sum of itself and the corresponding field of register A. Adjusts Carry.

B=B+B fs - Sum of B and B into B
----

fs = A	opcode:	C5
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Aa5
	cycles:	3 + d

Double the specified fs field of register B. Adjusts Carry.

B=B+C fs - Sum of B and C into B
----

fs = A	opcode:	C1
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Aa1
	cycles:	3 + d

Set the specified fs field of register B to the sum of itself and the corresponding field of register C. Adjusts Carry.

B=B-1 fs - Decrement B
----

fs = A	opcode:	CD
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	AaD
	cycles:	3 + d

Decrement the specified fs field of register B by one. Adjusts Carry.

```
B=B-A  fs   -  B minus A into B
---------
fs = A                        opcode:  E8
                              cycles:   7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  Ba8
                              cycles:   3 + d
```

Set the specified fs field of register B to the difference between itself and the corresponding field of register A. Adjusts Carry.

```
B=B-C  fs   -  B minus C into B
---------
fs = A                        opcode:  E1
                              cycles:   7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  Ba1
                              cycles:   3 + d
```

Set the specified fs field of register B to the difference between itself and the corresponding field of register C. Adjusts Carry.

```
B=C    fs   -  Copy C to B
---------
fs = A                        opcode:  D5
                              cycles:   7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  Ab5
                              cycles:   3 + d
```

Copy the fs field of register C into the corresponding field of register B. Carry is not affected.

B=C-B fs - C minus B into B
----

fs = A	opcode:	ED
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	BaD
	cycles:	3 + d

Set the specified fs field of register B to the inverse difference between itself and the corresponding field of register C. Adjusts Carry.

BAEX fs - Exchange Registers B and A
----

fs = A	opcode:	DC
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	AbC
	cycles:	3 + d

Exchange the fs fields of registers of B and A. Carry is not affected.

BCEX fs - Exchange Registers B and C
----

fs = A	opcode:	DD
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	AbD
	cycles:	3 + d

Exchange the fs fields of registers of B and C. Carry is not affected.

BSL fs - B Shift Left

----

fs = A	opcode: F1 cycles: 7
fs = (P,WP,XS,X,S,M,B,W)	opcode: Bb1 cycles: 3 + d

Shift the contents of the specified fs field of register B left one nibble, without affecting the rest of the register. The nibble shifted off the left end of the field is lost. The new low-order nibble of the field is zero. The Sticky Bit (SB) is not affected.

BSLC - B Shift Left Circular

----

opcode: 811
cycles: 21

Circular shift register B left one nibble. Operates on all 16 digits. The Sticky Bit (SB) is not affected.

BSR fs - B Shift Right

----

fs = A	opcode: F5 cycles: 7
fs = (P,WP,XS,X,S,M,B,W)	opcode: Bb5 cycles: 3 + d

Shift the contents of the specified fs field of register B right one nibble, without affecting the rest of the register. The nibble shifted off the right end of the field is lost, but the Sticky Bit (SB) is set if the nibble was non-zero. The new high-order nibble of the field is zero.

BSRB - B Shift Right Bit
----

opcode: 81D
cycles: 20

Shift register B right one bit. Operates on all 16 digits. The bit shifted off the end is lost, but the Sticky Bit (SB) is set if it was non-zero. The new high-order bit of the register is zero.

BSRC - B Shift Right Circular
----

opcode: 815
cycles: 21

Circular shift register B right one nibble. Operates on all 16 digits. The Sticky Bit (SB) is set if the nibble shifted from low-order around to high-order position was non-zero.

BUSCC - Bus Command "C"
----

opcode: 80B
cycles: 6

Enters the HP-71 bus command "C" onto the system bus (this command is reserved for later use). No other operation is performed. See the "HP-71 Hardware Specification" for more information.

C+P+1 - Increment C by One Plus P Pointer
----

opcode: 809
cycles: 8

The A field of the C register is incremented by one plus the value of the P pointer. This instruction is always executed in HEX mode. Adjusts Carry.

C=-C fs - Two's complement of C into C

----

fs = A	opcode:	FA
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	BbA
	cycles:	3 + d

Complement the specified fs field of C. Complement is two's complement if in HEX mode, ten's complement if in DEC mode. Carry is set if the field is not zero, else Carry is cleared.

C=-C-1 fs - One's complement of C into C

----

fs = A	opcode:	FE
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	BbE
	cycles:	3 + d

Perform a one's complement on the specified fs field of C. Carry is always cleared.

C=0 fs - Set C equal to 0

----

fs = A	opcode:	D2
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Ab2
	cycles:	3 + d

Set the specified fs field of C to zero. Carry is not affected.

C=A   fs   - Copy A to C
----

fs = A	opcode:	D6
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Ab6
	cycles:	3 + d

Copy the fs field of register A into the corresponding field of register C. Carry is not affected.

C=A-C fs   - A minus C into C
----

fs = A	opcode:	EE
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	BaE
	cycles:	3 + d

Set the specified fs field of register C to the inverse difference between itself and the corresponding field of register A. Adjusts Carry.

C=B   fs   - Copy B to C
----

fs = A	opcode:	D9
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Ab9
	cycles:	3 + d

Copy the fs field of register B into the corresponding field of register C. Carry is not affected.

```
C=C!A  fs   -  C OR A into C
---------
fs = A                          opcode:  0EFA
                                cycles:    4 + d

fs = (P,WP,XS,X,S,M,B,W)        opcode:  0EaA
                                cycles:    4 + d
```

Set the fs field of register C to its logical OR with the corresponding field of register A. Carry is not affected.

```
C=C!B  fs   -  C OR B into C
---------
fs = A                          opcode:  0EFD
                                cycles:    4 + d

fs = (P,WP,XS,X,S,M,B,W)        opcode:  0EaD
                                cycles:    4 + d
```

Set the fs field of register C to its logical OR with the corresponding field of register B. Carry is not affected.

```
C=C!D  fs   -  C OR D into C
---------
fs = A                          opcode:  0EFF
                                cycles:    4 + d

fs = (P,WP,XS,X,S,M,B,W)        opcode:  0EaF
                                cycles:    4 + d
```

Set the fs field of register C to its logical OR with the corresponding field of register D. Carry is not affected.

C=C&A fs - C AND A into C

```
fs = A                        opcode:  0EF2
                              cycles:    4 + d

fs = (P,WP,XS,X,S,M,B,W)      opcode:  0Ea2
                              cycles:    4 + d
```

Set the fs field of register C to its logical AND with the corresponding field of register A. Carry is not affected.

C=C&B fs - C AND B into C

```
fs = A                        opcode:  0EF5
                              cycles:    4 + d

fs = (P,WP,XS,X,S,M,B,W)      opcode:  0Ea5
                              cycles:    4 + d
```

Set the fs field of register C to its logical AND with the corresponding field of register B. Carry is not affected.

C=C&D fs - C AND D into C

```
fs = A                        opcode:  0EF7
                              cycles:    4 + d

fs = (P,WP,XS,X,S,M,B,W)      opcode:  0Ea7
                              cycles:    4 + d
```

Set the fs field of register C to its logical AND with the corresponding field of register D. Carry is not affected.

```
C=C+1  fs  -  Increment C
---------
fs = A                        opcode:  E6
                              cycles:   7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  Ba6
                              cycles:   3 + d
```

Increment the specified fs field of register C by one. Adjusts Carry.

```
C=C+A  fs   - Sum of C and A into C
---------
fs = A                        opcode:  C2
                              cycles:   7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  Aa2
                              cycles:   3 + d
```

Set the specified fs field of register C to the sum of itself and the corresponding field of register A. Adjusts Carry.

```
C=C+B  fs  -  Sum of C and B into C
---------
fs = A                        opcode:  C9
                              cycles:   7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  Aa9
                              cycles:   3 + d
```

Set the specified fs field of register C to the sum of itself and the corresponding field of register B. Adjusts Carry.

```
C=C+C  fs   - Sum of C and C into C
---------
fs = A                          opcode:  C6
                                cycles:    7

fs = (P,WP,XS,X,S,M,B,W)        opcode:  Aa6
                                cycles:    3 + d
```

Double the specified fs field of register C. Adjusts Carry.

```
C=C+D  fs   -  Sum of C and D into C
---------
fs = A                          opcode:  CB
                                cycles:    7

fs = (P,WP,XS,X,S,M,B,W)        opcode:  AaB
                                cycles:    3 + d
```

Set the specified fs field of register C to the sum of itself and the corresponding field of register D. Adjusts Carry.

```
C=C-1  fs   -  Decrement C
---------
fs = A                          opcode:  CE
                                cycles:    7

fs = (P,WP,XS,X,S,M,B,W)        opcode:  AaE
                                cycles:    3 + d
```

Decrement the specified fs field of register C by one. Adjusts Carry.

C=C-A fs - C minus A into C

```
fs = A                      opcode:  E2
                            cycles:    7

fs = (P,WP,XS,X,S,M,B,W)    opcode:  Ba2
                            cycles:    3 + d
```

Set the specified fs field of register C to the difference between itself and the corresponding field of register A. Adjusts Carry.

C=C-B fs - C minus B into C

```
fs = A                      opcode:  E9
                            cycles:    7

fs = (P,WP,XS,X,S,M,B,W)    opcode:  Ba9
                            cycles:    3 + d
```

Set the specified fs field of register C to the difference between itself and the corresponding field of register B. Adjusts Carry.

C=C-D fs - C minus D into C

```
fs = A                      opcode:  EB
                            cycles:    7

fs = (P,WP,XS,X,S,M,B,W)    opcode:  BaB
                            cycles:    3 + d
```

Set the specified fs field of register C to the difference between itself and the corresponding field of register D. Adjusts Carry.

C=D fs - Copy D to C
----

fs = A	opcode: DB cycles: 7
fs = (P,WP,XS,X,S,M,B,W)	opcode: AbB cycles: 3 + d

Copy the fs field of register D into the corresponding field of register C. Carry is not affected.

C=DAT0 fsd - Load C from memory
----

fs = A	opcode: 146 cycles: 18
fs = B	opcode: 14E cycles: 15
fs = (P,WP,XS,X,S,M,W)	opcode: 156a cycles: 17 + d
fs = d	opcode: 15Ex (x=d-1) cycles: 16 + d

The amount of data (d nibbles) specified by fsd will be transferred from the memory address pointed to by D0 into the specified field of register C. The lowest-addressed nibble will be transferred into the lowest-order nibble of the register field, proceeding toward the higher-order nibbles. If fs = d, d nibbles are transferred into the register starting at nibble 0. See the section on "Loading Data From Memory" earlier in this chapter.

C=DAT1 fsd - Load C from memory
----

fs = A	opcode:	147
	cycles:	18
fs = B	opcode:	14F
	cycles:	15
fs = (P,WP,XS,X,S,M,W)	opcode:	157a
	cycles:	17 + d
fs = d	opcode:	15Fx (x=d-1)
	cycles:	16 + d

The amount of data (d nibbles) specified by fsd will be transferred from the memory address pointed to by D1 into the specified field of register C. The lowest-addressed nibble will be transferred into the lowest-order nibble of the register field, proceeding toward the higher-order nibbles. If fs = d, d nibbles are transferred into the register starting at nibble 0. See the section on "Loading Data From Memory" earlier in this chapter.

C=ID - Request chip ID
----

opcode: 806
cycles: 11

The chip which has its DAISY-IN line high and its configuration flag low will send its 5 nibble ID register to the system bus which will be loaded into the low-order 5 nibbles (A field) of the C register. See the "HP-71 Hardware Specification" for more information.

C=IN - Load C with IN
----

opcode: 803
cycles: 7

Load the low-order 4 nibbles of the C register with the contents of the Input register. See the "HP-71 Hardware Specification" for more information.

C=P n - Copy P Pointer into Nibble n of C
----

opcode: 80Cn
cycles: 6

Copy P pointer into C register at digit position specified by n.

C=R0 - Copy R0 to C
----

opcode: 118
cycles: 19

The contents of the scratch register R0 is copied to the working register C.

C=R1 - Copy R1 to C
----

opcode: 119
cycles: 19

The contents of the scratch register R1 is copied to the working register C.

C=R2 - Copy R2 to C
----

opcode: 11A
cycles: 19

The contents of the scratch register R2 is copied to the working register C.

C=R3 - Copy R3 to C
----

opcode: 11B
cycles: 19

The contents of the scratch register R3 is copied to the working register C.

C=R4 - Copy R4 to C
----

opcode: 11C
cycles: 19

The contents of the scratch register R4 is copied to the working register C.

C=RSTK - Pop stack to C
----

opcode: 07
cycles: 8

Pop the top-most address off of the hardware return stack, placing the address in the lower 5 nibbles (A field) of register C. The high-order nibbles of C are unchanged. As the address is popped from the return stack, a zero address is inserted at the bottom of the stack. Compare with the RTN mnemonic.

C=ST - Status to C
----

opcode: 09
cycles: 6

Copy the low-order 12 bits of the status register into the low-order 12 bits (X field) of the C register.

CAEX fs - Exchange Registers C and A
----

fs = A opcode: DE
cycles: 7

fs = (P,WP,XS,X,S,M,B,W) opcode: AbE
cycles: 3 + d

Exchange the fs fields of registers of C and A. Carry is not affected.

CBEX fs - Exchange Registers C and B
----

fs = A opcode: DD
cycles: 7

fs = (P,WP,XS,X,S,M,B,W) opcode: AbD
cycles: 3 + d

Exchange the fs fields of registers of C and B. Carry is not affected.

CD0EX - Exchange C and D0 (nibs 0-4)
----

opcode: 136
cycles: 8

Exchange the A field of register C with Data pointer D0. Carry is not affected.

CD0XS - Exchange C and D0 short (nibs 0-3)
----

opcode: 13E
cycles: 7

Exchange the lower 4 nibbles of C with the lower 4 nibbles of Data pointer D0. Carry is not affected.

CD1EX - Exchange C and D1 (nibs 0-4)
----

opcode: 137
cycles: 8

Exchange the A field of register C with Data pointer D1. Carry is not affected.

CD1XS - Exchange C and D1 short (nibs 0-3)
----

opcode: 13F
cycles: 7

Exchange the lower 4 nibbles of C with the lower 4 nibbles of Data pointer D1. Carry is not affected.

CDEX fs - Exchange Registers C and D

```
fs = A                          opcode:  DF
                                cycles:   7

fs = (P,WP,XS,X,S,M,B,W)        opcode:  AbF
                                cycles:   3 + d
```

Exchange the fs fields of registers of C and D. Carry is not affected.

CLRHST - Clear Hardware Status bits

```
                                opcode:  82F
                                cycles:   3
```

Clears the 4 Hardware Status bits XM, SB, SR and MP. Note that the opcode is actually 82x, where x is merely a mask for which Hardware Status bits to clear, as follows:

```
   bit 0 -  External Module Missing bit (see XM=0 mnemonic)
   bit 1 -  Sticky Bit                  (see SB=0 mnemonic)
   bit 2 -  Service Request bit         (see SR=0 mnemonic)
   bit 3 -  Module Pulled bit           (see MP=0 mnemonic)
```

For example opcode 829 clears XM and MP. Although there is no mnemonic for this, the opcode can be inserted into the code by using, for example, NIBHEX 829.

CLRST - Clear Program Status

```
                                opcode:  08
                                cycles:   6
```

Clear the low-order 12 bits (S0 through S11) of the Program Status register ST.

CONFIG - Configure
----

opcode: 805
cycles: 11

Copy the low-order 5 nibbles (A field) of the C register into the Configuration register of the chip which has its DAISY-IN line high and its configuration flag low. See the "HP-71 Hardware Specification" for information.

CPEX n - Exchange Nibble n of C With P Pointer
----

opcode: 80Fn
cycles: 6

Exchange the P pointer with digit n of the C register.

CR0EX - Exchange C and R0
----

opcode: 128
cycles: 19

Exchange the contents of the working register C and the scratch register R0.

CR1EX - Exchange C and R1
----

opcode: 129
cycles: 19

Exchange the contents of the working register C and the scratch register R1.

CR2EX - Exchange C and R2
----

opcode: 12A
cycles: 19

Exchange the contents of the working register C and the scratch register R2.

CR3EX - Exchange C and R3
----

opcode: 12B
cycles: 19

Exchange the contents of the working register C and the scratch register R3.

CR4EX - Exchange C and R4
----

opcode: 12C
cycles: 19

Exchange the contents of the working register C and the scratch register R4.

CSL fs - C Shift Left
----

fs = A	opcode:	F2
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	Bb2
	cycles:	3 + d

Shift the contents of the specified fs field of register C left one nibble, without affecting the rest of the register. The nibble shifted off the left end of the field is lost. The new low-order nibble of the field is zero. The Sticky Bit (SB) is not affected.

CSLC - C Shift Left Circular

----

opcode: 812
cycles: 21

Circular shift register C left one nibble. Operates on all 16 digits. The Sticky Bit (SB) is not affected.

CSR fs - C Shift Right

----

fs = A opcode: F6
cycles: 7

fs = (P,WP,XS,X,S,M,B,W) opcode: Bb6
cycles: 3 + d

Shift the contents of the specified fs field of register C right one nibble, without affecting the rest of the register. The nibble shifted off the right end of the field is lost, but the Sticky Bit (SB) is set if the nibble was non-zero. The new high-order nibble of the field is zero.

CSRB - C Shift Right Bit

----

opcode: 81E
cycles: 20

Shift register C right one bit. Operates on all 16 digits. The bit shifted off the end is lost, but the Sticky Bit (SB) is set if it was non-zero. The new high-order bit of the register is zero.

CSRC - C Shift Right Circular
----

opcode: 816
cycles: 21

Circular shift register C right one nibble. Operates on all 16 digits. The Sticky Bit (SB) is set if the nibble shifted from low-order around to high-order position was non-zero.

CSTEX - Exchange Status with C register
----

opcode: 0B
cycles: 6

Exchange the low-order 12 bits (S0 through S11) of the Program Status register ST with the low-order 12 bits (X field) of the C register.

D0=(2) nn - Load 2 Nibbles Into D0
----

opcode: 19nn
cycles: 4

Load the low-order two nibbles of D0 with nn. The upper nibbles of D0 remain unchanged. Any overflow is ignored by the assembler. The assembled digits of nn are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory" earlier in this chapter.

D0=(4) nnnn - Load 4 Nibbles Into D0

opcode: 1Annnn
cycles: 6

Load the low-order four nibbles of D0 with nnnn. The upper nibble of D0 remains unchanged. Any overflow is ignored by the assembler. The assembled digits of nnnn are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory" earlier in this chapter.

D0=(5) nnnnn - Load 5 Nibbles Into D0

opcode: 1Bnnnnn
cycles: 7

Load all five nibbles of D0 with nnnnn. Any overflow is ignored by the assembler. The assembled digits of nnnnn are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory" earlier in this chapter.

D0=A - Copy A to D0 (nibs 0-4)

opcode: 130
cycles: 8

The A field of register A is copied into Data pointer register D0. Carry is not affected.

```
D0=AS      - Copy A to D0 short (nibs 0-3)
---------
                              opcode:  138
                              cycles:    7
```

The lower 4 nibbles of A are copied into the lower 4 nibbles of Data pointer register D0. Carry is not affected.

```
D0=C       - Copy C to D0 (nibs 0-4)
---------
                              opcode:  134
                              cycles:    8
```

The A field of register C is copied into Data pointer register D0. Carry is not affected.

```
D0=CS      - Copy C to D0 short (nibs 0-3)
---------
                              opcode:  13C
                              cycles:    7
```

The lower 4 nibbles of C are copied into the lower 4 nibbles of Data pointer register D0. Carry is not affected.

```
D0=D0+ n   - Add n to D0 (1<=n<=16)
---------
                              opcode:  16x (x=n-1)
                              cycles:    7
```

Increment D0 by n. This instruction is always executed in HEX mode. Adjusts Carry.

D0=D0- n - Subtract n from D0 (1<=n<=16)
----

opcode: 18x (x=n-1)
cycles: 7

Decrement D0 by n. This instruction is always executed in HEX mode. Adjusts Carry.

D0=HEX hh - Load D0 with hex constant hh
----

opcode: 19hh
cycles: 4

Load the low-order two nibbles of D0 with the hex constant hh. The upper nibbles of D0 remain unchanged. The digits of hh are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory" earlier in this chapter.

D0=HEX hhhh - Load D0 with hex constant hhhh
----

opcode: 1Ahhhh
cycles: 6

Load the low-order four nibbles of D0 with the hex constant hhhh. The upper nibble of D0 remains unchanged. The digits of hhhh are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory" earlier in this chapter.

D0=HEX hhhhh - Load D0 with hex constant hhhhh
----

opcode: 1Bhhhhh
cycles: 7

Load all five nibbles of D0 with the hex constant hhhhh. The digits of hhhhh are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory" earlier in this chapter.

D1=(2) nn - Load 2 Nibbles Into D1
----

opcode: 1Dnn
cycles: 4

Load the low-order two nibbles of D1 with nn. The upper nibbles of D1 remain unchanged. Any overflow is ignored by the assembler. The assembled digits of nn are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory" earlier in this chapter.

D1=(4) nnnn - Load 4 Nibbles Into D1
----

opcode: 1Ennnn
cycles: 6

Load the low-order four nibbles of D1 with nnnn. The upper nibble of D1 remains unchanged. Any overflow is ignored by the assembler. The assembled digits of nnnn are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory" earlier in this chapter.

D1=(5) nnnnn - Load 5 Nibbles Into D1
----

opcode: 1Fnnnnn
cycles: 7

Load all five nibbles of D1 with nnnnn. Any overflow is ignored by the assembler. The assembled digits of nnnnn are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory" earlier in this chapter.

D1=A - Copy A to D1 (nibs 0-4)
----

opcode: 131
cycles: 8

The A field of register A is copied into Data pointer register D1. Carry is not affected.

D1=AS - Copy A to D1 short (nibs 0-3)
----

opcode: 139
cycles: 7

The lower 4 nibbles of A are copied into the lower 4 nibbles of Data pointer register D1. Carry is not affected.

D1=C - Copy C to D1 (nibs 0-4)
----

opcode: 135
cycles: 8

The A field of register C is copied into Data pointer register D1. Carry is not affected.

D1=CS - Copy C to D1 short (nibs 0-3)
----

opcode: 13D
cycles: 7

The lower 4 nibbles of C are copied into the lower 4 nibbles of Data pointer register D1. Carry is not affected.

D1=D1+ n - Add n to D1 (1<=n<=16)
----

opcode: 17x (x=n-1)
cycles: 7

Increment D1 by n. This instruction is always executed in HEX mode. Adjusts Carry.

D1=D1- n - Subtract n from D1 (1<=n<=16)
----

opcode: 1Cx (x=n-1)
cycles: 7

Decrement D1 by n. This instruction is always executed in HEX mode. Adjusts Carry.

D1=HEX hh - Load D1 with hex constant hh
----

opcode: 1Dhh
cycles: 4

Load the low-order two nibbles of D1 with the hex constant hh. The upper nibbles of D1 remain unchanged. The digits of hh are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory" earlier in this chapter.

D1=HEX hhhh - Load D1 with hex constant hhhh

opcode: 1Ehhhh
cycles: 6

Load the low-order four nibbles of D1 with the hex constant hhhh. The upper nibble of D1 remains unchanged. The digits of hhhh are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory" earlier in this chapter.

D1=HEX hhhhh - Load D1 with hex constant hhhhh

opcode: 1Fhhhhh
cycles: 7

Load all five nibbles of D1 with the hex constant hhhhh. The digits of hhhhh are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory" earlier in this chapter.

D=-D fs - Two's complement of D into D

fs = A — opcode: FB; cycles: 7

fs = (P,WP,XS,X,S,M,B,W) — opcode: BbB; cycles: 3 + d

Complement the specified fs field of D. Complement is two's complement if in HEX mode, ten's complement if in DEC mode. Carry is set if the field is not zero, else Carry is cleared.

```
D=-D-1 fs  -  One's complement of D into D
---------
fs = A                        opcode:  FF
                              cycles:   7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  BbF
                              cycles:   3 + d
```

Perform a one's complement on the specified fs field of D. Carry is always cleared.

```
D=0    fs  -  Set D equal to 0
---------
fs = A                        opcode:  D3
                              cycles:   7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  Ab3
                              cycles:   3 + d
```

Set the specified fs field of D to zero. Carry is not affected.

```
D=C    fs  -  Copy C to D
---------
fs = A                        opcode:  D7
                              cycles:   7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  Ab7
                              cycles:   3 + d
```

Copy the fs field of register C into the corresponding field of register D. Carry is not affected.

```
D=C-D  fs   -  C minus D into D
---------
fs = A                        opcode:  EF
                              cycles:    7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  BaF
                              cycles:    3 + d
```

Set the specified fs field of register D to the inverse difference between itself and the corresponding field of register C. Adjusts Carry.

```
D=D!C  fs   -  D OR C into D
---------
fs = A                        opcode:  0EFB
                              cycles:    4 + d

fs = (P,WP,XS,X,S,M,B,W)      opcode:  0EaB
                              cycles:    4 + d
```

Set the fs field of register D to its logical OR with the corresponding field of register C. Carry is not affected.

```
D=D&C  fs   -  D AND C into D
---------
fs = A                        opcode:  0EF3
                              cycles:    4 + d

fs = (P,WP,XS,X,S,M,B,W)      opcode:  0Ea3
                              cycles:    4 + d
```

Set the fs field of register D to its logical AND with the corresponding field of register C. Carry is not affected.

```
D=D+1  fs   -  Increment D
---------
fs = A                        opcode:  E7
                              cycles:    7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  Ba7
                              cycles:    3 + d
```

Increment the specified fs field of register D by one. Adjusts Carry.

```
D=D+C  fs   -  Sum of D and C into D
---------
fs = A                        opcode:  C3
                              cycles:    7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  Aa3
                              cycles:    3 + d
```

Set the specified fs field of register D to the sum of itself and the corresponding field of register C. Adjusts Carry.

```
D=D+D  fs   -  Sum of D and D into D

---------
fs = A                        opcode:  C7
                              cycles:    7

fs = (P,WP,XS,X,S,M,B,W)      opcode:  Aa7
                              cycles:    3 + d
```

Double the specified fs field of register D. Adjusts Carry.

D=D-1 fs - Decrement D
----

fs = A   opcode: CF
                                cycles: 7

fs = (P,WP,XS,X,S,M,B,W)   opcode: AaF
                                cycles: 3 + d

Decrement the specified fs field of register D by one. Adjusts Carry.

D=D-C fs - D minus C into D
----

fs = A   opcode: E3
                                cycles: 7

fs = (P,WP,XS,X,S,M,B,W)   opcode: Ba3
                                cycles: 3 + d

Set the specified fs field of register D to the difference between itself and the corresponding field of register C. Adjusts Carry.

DAT0=A fsd - Store into memory from A
----

fs = A   opcode: 140
                                cycles: 17

fs = B   opcode: 148
                                cycles: 14

fs = (P,WP,XS,X,S,M,W)   opcode: 150a
                                cycles: 16 + d

fs = d   opcode: 158x (x=d-1)
                                cycles: 15 + d

The amount of data (d nibbles) specified by fsd will be written to the memory address pointed to by D0 from the specified field of register A. The lowest-order nibble of the register field will be written to the lowest-addressed nibble of memory, proceeding toward the higher-order nibbles. If fs = d, d nibbles are written to memory starting from nibble 0 of the register. See the section on "Storing Data Into Memory" earlier in this chapter.

DAT0=C fsd - Store into memory from C

----

fs = A	opcode:	144
	cycles:	17
fs = B	opcode:	14C
	cycles:	14
fs = (P,WP,XS,X,S,M,W)	opcode:	154a
	cycles:	16 + d
fs = d	opcode:	15Cx (x=d-1)
	cycles:	15 + d

The amount of data (d nibbles) specified by fsd will be written to the memory address pointed to by D0 from the specified field of register C. The lowest-order nibble of the register field will be written to the lowest-addressed nibble of memory, proceeding toward the higher-order nibbles. If fs = d, d nibbles are written to memory starting from nibble 0 of the register. See the section on "Storing Data Into Memory" earlier in this chapter.

DAT1=A fs - Store into memory from A

----

fs = A	opcode:	141
	cycles:	17
fs = B	opcode:	149
	cycles:	14
fs = (P,WP,XS,X,S,M,W)	opcode:	151a
	cycles:	16 + d
fs = d	opcode:	159x (x=d-1)
	cycles:	15 + d

The amount of data (d nibbles) specified by fsd will be written to the memory address pointed to by D1 from the specified field of register A. The lowest-order nibble of the register field will be written to the lowest-addressed nibble of memory, proceeding toward the higher-order nibbles. If fs = d, d nibbles are written to memory starting from nibble 0 of the register. See the section on "Storing Data Into Memory" earlier in this chapter.

DAT1=C fsd - Store into memory from C
----

fs = A	opcode:	145
	cycles:	17
fs = B	opcode:	14D
	cycles:	14
fs = (P,WP,XS,X,S,M,W)	opcode:	155a
	cycles:	16 + d
fs = d	opcode:	15Dx (x=d-1)
	cycles:	15 + d

The amount of data (d nibbles) specified by fsd will be written to the memory address pointed to by D1 from the specified field of register C. The lowest-order nibble of the register field will be written to the lowest-addressed nibble of memory, proceeding toward the higher-order nibbles. If fs = d, d nibbles are written to memory starting from nibble 0 of the register. See the section on "Storing Data Into Memory" earlier in this chapter.

DCEX fs - Exchange Registers D and C
----

fs = A	opcode:	DF
	cycles:	7
fs = (P,WP,XS,X,S,M,B,W)	opcode:	AbF
	cycles:	3 + d

Exchange the fs fields of registers of D and C. Carry is not affected.

DSL fs - D Shift Left

```
fs = A                         opcode:  F3
                               cycles:    7

fs = (P,WP,XS,X,S,M,B,W)       opcode:  Bb3
                               cycles:    3 + d
```

Shift the contents of the specified fs field of register D left one nibble, without affecting the rest of the register. The nibble shifted off the left end of the field is lost. The new low-order nibble of the field is zero. The Sticky Bit (SB) is not affected.

DSLC - D Shift Left Circular

```
                               opcode:  813
                               cycles:   21
```

Circular shift register D left one nibble. Operates on all 16 digits. The Sticky Bit (SB) is not affected.

DSR fs - D Shift Right

```
fs = A                         opcode:  F7
                               cycles:    7

fs = (P,WP,XS,X,S,M,B,W)       opcode:  Bb7
                               cycles:    3 + d
```

Shift the contents of the specified fs field of register D right one nibble, without affecting the rest of the register. The nibble shifted off the right end of the field is lost, but the Sticky Bit (SB) is set if the nibble was non-zero. The new high-order nibble of the field is zero.

DSRB - D Shift Right Bit
----

opcode: 81F
cycles: 20

Shift register D right one bit. Operates on all 16 digits. The bit shifted off the end is lost, but the Sticky Bit (SB) is set if it was non-zero. The new high-order bit of the register is zero.

DSRC - D Shift Right Circular
----

opcode: 817
cycles: 21

Circular shift register D right one nibble. Operates on all 16 digits. The Sticky Bit (SB) is set if the nibble shifted from low-order around to high-order position was non-zero.

GOC label - Go relative on carry
----

opcode: 4aa
cycles: 10 (GO)
3 (NO)

Short relative jump to label if Carry is set. label must be in the range:

addr - 128 <= label <= addr + 127

where addr is the address of the second nibble of the opcode. The address offset aa is in two's complement form and is relative to addr.

GOLONG label - Go Long

opcode: 8Caaaa
cycles: 14

Long relative jump to label unconditionally. label must be in the range:

addr - 32768 <= label <= addr + 32767

where addr is the address of the third nibble of the opcode. The address offset aaaa is in two's complement form and is relative to addr.

GONC label - Go relative on no carry

opcode: 5aa
cycles: 10 (GO)
3 (NO)

Short relative jump to label if Carry is clear. label must be in the range:

addr - 128 <= label <= addr + 127

where addr is the address of the second nibble of the opcode. The address offset aa is in two's complement form and is relative to addr.

GOSBVL label - Gosub very long to label

opcode: 8Faaaaa
cycles: 15

Absolute subroutine jump to aaaaa, which is the absolute address of label. See the GOSUB mnemonic.

GOSUB label - Gosub to label
----

opcode: 7aaa
cycles: 12

Relative subroutine jump to label. label must be in the range:

addr - 2048 <= label <= addr + 2047

where addr is the starting address of the next instruction. The address offset aaa is in two's complement form and is relative to addr.

As with all subroutine jumps, the address (addr) of the instruction following the gosub opcode is pushed onto the hardware return stack, so that when a corresponding return is executed, control resumes with the instruction at address addr.

As the return address is pushed onto the return stack, the bottom-most address on the stack is discarded. Therefore, the return stack always contains 8 addresses, and if pushes exceed pops by 8 levels, the bottom-most return addresses are lost. Since the interrupt system requires one level to process interrupts, only 7 levels of the return stack can be used by code which must execute when interrupts are enabled. See the RTN mnemonic for further information.

GOSUBL label - Gosub long to label
----

opcode: 8Eaaaa
cycles: 15

Long relative subroutine jump to label. label must be in the range:

addr - 32768 <= label <= addr + 32767

where addr is the starting address of the next instruction. The address offset aaaa is in two's complement form and is relative to addr. See the GOSUB mnemonic.

GOTO label - Jump relative
----

opcode: 6aaa
cycles: 11

Relative jump to label unconditionally. label must be in the range:

addr - 2048 <= label <= addr + 2047

where addr is the address of the second nibble of the opcode. The address offset aaa is in two's complement form and is relative to addr.

GOVLNG label - Jump very long
----

opcode: 8Daaaaa
cycles: 14

Unconditional jump to aaaaa, which is the absolute address of label.

GOYES label - Jump if Test is True
----

opcode: yy
cycles: included in the accompaning Test mnemonic cycle time.

GOYES is a mnemonic to specify part of a CPU test opcode. GOYES must always follow a test mnemonic. If the condition of the test is met, a jump is performed to label with Carry set. label must be in the range

addr - 128 <= label <= addr + 127

where addr is the starting address of the jump offset yy. If the test condition is not met, Carry is cleared and control passes to the next instruction. Compare with RTNYES.

INTOFF - Interrupt Off
----

opcode: 808F
cycles: 5

Disable the keyboard interrupt system. However, INTOFF does not disable the "ON" key or *INT line interrupts. See the "HP-71 Hardware Specification" for more information.

INTON - Interrupt On
----

opcode: 8080
cycles: 5

Enable the keyboard interrupt system. See the "HP-71 Hardware Specification" for more information.

LC(m) n..n - Load C with constant (1<=m<=6)
----

opcode: 3xn..n (x=m-1)
cycles: 3+m

Load m digits of the expression n..n to the C register beginning at the P pointer position, and proceeding toward higher-order nibbles, with the ability to wrap around the register. See the section on "Loading Data From Memory" earlier in this chapter.

LCASC \A..A\ - Load C with ASCII constant
----

opcode: 3mc..c
(m = 2*(# of chars)-1;
c..c = ASCII codes)
cycles: 3+2*(# of chars)

Load up to 8 ASCII characters to the C register beginning at the P pointer position, and proceeding toward higher-order nibbles, with the ability to wrap around the register. Each A represents an ASCII character. The ASCII characters are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory" earlier in this chapter.

LCHEX h..h - Load C with hex constant

opcode: 3nh..h (n=# of digits-1)
cycles: 4+n

Load up to 16 hex digits into the C register beginning at the P pointer position, and proceeding toward higher-order nibbles, with the ability to wrap around the register. The hex digits are stored in the opcode in reverse order so that when the instruction is executed the data will be loaded into the register with the intended orientation. See the section on "Loading Data From Memory" earlier in this chapter.

MP=0 - Clear Module Pulled bit (MP)

opcode: 828
cycles: 3

Clears the Module Pulled bit (MP) and pulls the Module Pulled Interrupt line low. See CLRHST mnemonic.

NOP3 - Three nibble No-op

```
opcode:  420
cycles:  10 (GO/RTNYES)
          3 (NO)
```

This mnemonic generates a GOC to the next instruction, effectively skipping three nibbles.

NOP4 - Four nibble No-op

```
opcode:  6300
cycles:   11
```

This mnemonic generates a GOTO to the next instruction, effectively skipping four nibbles.

NOP5 - Five nibble No-op

```
opcode:  64000
cycles:    11
```

This mnemonic generates a relative GOTO to +4 nibbles. The fifth nibble in the opcode is a place holder and is jumped over. The mnemonic effectively skips five nibbles.

OUT=C - Load 3 nibbles of OR

```
opcode:  801
cycles:    6
```

All nibbles of the Output register are loaded with the low-order three nibbles of C (X field).

OUT=CS - Load 1 nibble of OR
------

opcode: 800
cycles: 4

The least significant nibble of the Output register is loaded with the least significant nibble of the C register.

P=C n - Copy P pointer from C at Nibble n
----

opcode: 80Dn
cycles: 6

Copy nibble n of register C into the P pointer.

P=P+1 - Increment P Pointer
----

opcode: 0C
cycles: 3

Increment the P pointer. If P is incremented past F it will automatically wrap around to 0. This instruction is always executed in HEX mode. Adjusts carry.

P=P-1 - Decrement P Pointer
----

opcode: 0D
cycles: 3

Decrement the P pointer. If P is decremented past 0 it automatically wraps around to F. This instruction is always executed in HEX mode. Adjusts Carry.

P= n - Set P Pointer to n
----

opcode: 2n
cycles: 2

Set the P pointer to n.

R0=A - Copy A to register R0
----

opcode: 100
cycles: 19

The contents of the working register A is copied to the scratch register R0.

R0=C - Copy C to register R0
----

opcode: 108
cycles: 19

The contents of the working register C is copied to the scratch register R0.

R1=A - Copy A to register R1
----

opcode: 101
cycles: 19

The contents of the working register A is copied to the scratch register R1.

R1=C - Copy C to register R1
----

opcode: 109
cycles: 19

The contents of the working register C is copied to the scratch register R1.

R2=A - Copy A to register R2
----

opcode: 102
cycles: 19

The contents of the working register A is copied to the scratch register R2.

R2=C - Copy C to register R2
----

opcode: 10A
cycles: 19

The contents of the working register C is copied to the scratch register R2.

R3=A - Copy A to register R3
----

opcode: 103
cycles: 19

The contents of the working register A is copied to the scratch register R3.

R3=C - Copy C to register R3
----

opcode: 10B
cycles: 19

The contents of the working register C is copied to the scratch register R3.

R4=A - Copy A to register R4
----

opcode: 104
cycles: 19

The contents of the working register A is copied to the scratch register R4.

R4=C - Copy C to register R4
----

opcode: 10C
cycles: 19

The contents of the working register C is copied to the scratch register R4.

RESET - System reset
----

opcode: 80A
cycles: 6

The System Reset Bus Command is issued with all chips performing a local reset. The reset function will vary according to the chip type. See the "HP-71 Hardware Specification" for more information.

RSTK=C - Push C to Return Stack
----

opcode: 06
cycles: 8

Push the low-order 5 nibbles (A field) of the C register onto the Return Stack. The C register remains unchanged. See the GOSUB mnemonic.

RTI - Return from interrupt
----

opcode: 0F
cycles: 9

Return and re-enable the interrupt system. See the RTN mnemonic, and also the "HP-71 Hardware Specification."

RTN - Return
----

opcode: 01
cycles: 9

Return control to the top address on the hardware return stack. The top address on the hardware return stack is popped off and placed in the program counter PC. As the address is popped off the stack, a zero address is inserted at the bottom of the stack.

Therefore the hardware return stack always contains 8 addresses, and if more pops (returns) than pushes (gosubs) are performed, zeros will be read off the stack. Such an attempt to "return" to address 0 results in a memory reset, since the memory reset code of the operating system resides at address 0.

RTNC -Return on carry

```
                              opcode:  400
                              cycles:   10 (RTN)
                                         3 (NO)
```

Return if Carry is set. See RTN mnemonic.

RTNCC - Return, clear carry

```
                              opcode:  03
                              cycles:    9
```

Return and clear Carry. See RTN mnemonic.

RTNNC - Return on no carry

```
                              opcode:  500
                              cycles:   10 (RTN)
                                         3 (NO)
```

Return if Carry is not set. See RTN mnemonic.

RTNSC - Return, set carry

```
                              opcode:  02
                              cycles:    9
```

Return and set Carry. See RTN mnemonic.

RTNSXM - Return, set External Module Missing bit (XM)
----

opcode: 00
cycles: 9

Return and set the External Module Missing bit (XM). Since the opcode is zero, this mnemonic is executed on a jump to a non-existent memory device. See the "HP-71 Hardware Specification" for more information. See also the RTN mnemonic.

RTNYES - Return if Test is True
----

opcode: 00
cycles: included in the accompaning mnemonic cycle time.

RTNYES is a mnemonic to specify part of a CPU test opcode. RTNYES must always follow a test mnemonic. If the test condition is met, Carry is set and a return is executed. If the test condition is not met, control passes to the instruction following the RTNYES. Compare with the RTN and GOYES mnemonics.

SB=0 - Clear Sticky Bit (SB)
----

opcode: 822
cycles: 3

Clear the Sticky Bit (SB). See CLRHST mnemonic.

SETDEC - Set decimal
----

opcode: 05
cycles: 3

Set CPU arithmetic mode to decimal.

SETHEX - Set hexadecimal mode
----

opcode: 04
cycles: 3

Set CPU arithmetic mode to hexadecimal.

SHUTDN - System Shutdown
----

opcode: 807
cycles: 5

When this mnemonic is executed the CPU sends out the Shutdown Bus Command and stops its clock. Issuing the SHUTDN command with the least significant bit of the output register equal to 0 (i.e., if bit0=0) will immediately cause a cold start -- the PC is set to 0 and the CPU is not halted. This action is to insure that the ON key will be able to wake up the CPU. See the "HP-71 Hardware Specification" for more information.

SR=0 - Clear Service Request bit (SR)
----

opcode: 824
cycles: 3

Clear the Service Request bit (SR). See the CLRHST mnemonic.

SREQ?   - Service Request
----

opcode: 80E
cycles: 7

This mnemonic sets the Service Request bit (SR) if any chip on the system bus requests service. When it is executed, a Service Request Bus Command is issued on the system bus to poll all chips for a Service Request. If any chip requests service, a bus line will be pulled high during the next strobe following the Service Request Bus Command. This value of the bus will be latched into the least significant nibble of the C register. The bus line pulled high determines the device type according to the following table.

Bit	Device
3	Unused
2	Card Reader
1	HP-IL Mailbox
0	Display Driver (timer)

If any bus line is high, the Service Request bit (SR) will be set. See the "HP-71 Hardware Specification" for more information. See also the ?SREQ and SR=0 mnemonics.

ST=0   n   - Clear Program Status bit n
----

opcode: 84n
cycles: 4

Clear the Program Status bit selected by n.

ST=1   n   - Set Program Status bit n
----

opcode: 85n
cycles: 4

Set the Program Status bit selected by n.

```
ST=C      - C to Status
---------
                              opcode:  0A
                              cycles:   6
```

Copy the low-order 12 bits of the C register (X field) into the low-order 12 bits of the Status register.

```
UNCNFG    - Unconfigure
---------
                              opcode:  804
                              cycles:   12
```

Load the low-order 5 nibbles (A field) of the C register into the data address register of each peripheral chip, with the device addressed by the C register unconfiguring. See the "HP-71 Hardware Specification" for more information.

```
XM=0      - Clear External Module Missing bit (XM)
---------
                              opcode:  821
                              cycles:   3
```

Clear the External Module Missing bit (XM). This hardware status bit is set by the RTNSXM mnemonic. See the CLRHST mnemonic.

DISPLAY DRIVER CHIP — CHAPTER 5

This chapter contains a description of the 1LF3 display driver chip. The 1LF3 is designed to support the 1LF2 CPU and future processors that operate on the HP-71 bus. The display driver chip is divided into 4 functional areas:

1) An 8-way multiplexed Liquid Crystal Display (LCD) driver capable of functioning with other 1LF3's in a multi-chip display driver system.

2) 1K nibbles of RAM.

3) A 24 bit, quartz-crystal controlled timer.

4) A low battery indicator.

5.1 Pin Designations

The display driver's external pins are as follows:

PIN	DESCRIPTION
VDD	Power supply.
GND	System ground.
BUS[0:3]	System bus.
*STR	*STROBE
*CD	*CONTROL-DATA
OD	OUTPUT DISABLE - When high the bus is tristated; pulled low by an internal resistor.
CB(0,1)	CONFIGURATION BITs - 2 bits used for address configuration.
*CLK	Display-Timer clock; driven by MASTER.

*DON	*DISPLAY ON - Driven by MASTER; active low.
VREF	LCD voltage reference; driven by MASTER.
C[1:40]	40 LCD column drivers.
R[1:8]	8 LCD row drivers on MASTER; 8 LCD column drivers on SLAVE.
OSC(A,B)	2 pins used for quartz crystal connection.

5.2 Bus Commands

The display driver is a hard configured chip with service request capability. It responds to the bus commands listed in section 2.1.2 with the following exceptions:

1	ID	Same as NOP
8	CONFIGURE	Same as NOP
9	UNCONFIGURE	Same as NOP
A	POLL	If the timer has timed out (MSB=1) BUS0 is pulled high during the next *STR low (see section 2.4).
C	BUSCC	Same as NOP
E	SHUTDOWN	If the timer has timed out or times out during shutdown, *CD is pulled low to wake up the CPU (see section 2.4).
F	RESET	Same as NOP

5.3 Addressing

The 1K nibble of RAM requires 10 bits of address space, leaving 10 bits of configuration address. The most significant 8 bits of the RAM configuration are hard programmed. The other 2 bits are specified by CB0 and CB1.

The display RAM and timer are configured together and require 8 bits of address space, leaving 12 bits of configuration address. The most significant 10 bits of the display-timer configuration are hard programmed and the other 2 bits are specified by CB0 and CB1.

The address space is mapped as shown below in hex. The reserved space contains no memory and cannot be used elsewhere in the system.

	SLAVE1	SLAVE2	MASTER
RAM (1024 nibbles)			
Starting addr =	2F400	2F800	2FC00
Ending addr =	2F7FF	2FBFF	2FFFF
Display RAM (96 nibbles)			
Starting addr =	2E100	2E200	2E300
Ending addr =	2E15F	2E2FF	2E35F
RESERVED (152 nibbles)			
Starting addr =	2E160	2E260	2E360
Ending addr =	2E1F7	2E2F7	2E3F7
Timer (6 nibbles - least significant first)			
Starting addr =	2E1F8	2E2F8	2E3F8
Ending addr =	2E1FD	2E2FD	2E3FD
Contrast Control Nibble (1 nibble - active only on MASTER)			
Addr location =	2E1FE	2E2FE	2E3FE
Display-Timer Control Nibble (1 nibble)			
Addr location =	2E1FF	2E2FF	2E3FF

5.4 Display Interface

The display driver chips function in one of 2 modes: MASTER or SLAVE. The MASTER has both CB1 and CB0 tied high by the PC board. It drives the timing signal *CLK, the control signal *DON, and the LCD voltage reference VREF. The MASTER's outputs R[1:8] drive the 8 rows of the LCD and it's outputs C[1:40] drive the 40 LCD columns on the right side of the display. The SLAVE chips accept *CLK, *DON, and *VREF as inputs. SLAVE1 has CB0 tied high and CB1 tied low and it's outputs R[1:8] and C[1:40] drive the 48 LCD columns in the center of the display. SLAVE2 has CB0 tied low and CB1 tied high and it's outputs R[1:8] and C[1:40] drive the 48 LCD columns on the left side of the display.

Each dot of the LCD has a corresponding bit in the display RAM. If that bit is a 1 and the display is on, the corresponding dot will be on.

16 nibbles of the MASTER's display RAM are allocated to row driver data used to control the row driver waveforms. The following data pattern is loaded by the operating system beginning at the hex address 2E350 of the MASTER.

Hex data (low addressed nibble first) = 8001400220041008

If the above data pattern has been loaded, the least significant bit of the low addressed nibble of the display RAM will correspond to the upper-left dot of the display; the next bit of that nibble will be mapped to just below that dot; and dot addresses will continue to increase down and to the right.

The data pattern shown is not the only valid pattern. For example, try entering : POKE '2E350','0180024004200810'. This pattern defines the rows in reverse order and will result in characters (and annunciators) being displayed upside-down. This can be fixed be either poking the correct pattern or by an INIT:1.

5.5 Contrast Control Nibble

A contrast control nibble is located at address 2E3FE of the MASTER. This nibble adjusts the LCD drive voltage, VREF and thus can be used to control the contrast of the display. The higher the value of the contrast control nibble the darker the dots appear. This nibble can be both read and written and has no effect if the chip is configured as a SLAVE.

5.6 Display-Timer Control Nibble

A display-timer control nibble is located at address 2EnFF of both MASTER and SLAVE chips. The bits of this control nibble are defined as follows. Note that some bits have different meanings for read and write.

```
BIT0  -  READ & WRITE : DISPLAY ON; MASTER only; 1=on.

BIT1  -  READ & WRITE : DISPLAY BLINK; MASTER only; 1=blink
                        if BIT0 is set.

BIT2  -  READ  : VERY LOW BATTERY INDICATOR; MASTER only;
                 1=very low.
         WRITE : DISPLAY TEST; WARNING !!! DO NOT SET THIS BIT
                 ON MASTER, IT MAY FORCE DISPLAY OUT OF SYNC.
                 MASTER only; When set *CLK is synchronized
                 with *STR (for testing).
```

BIT3 - READ : LOW BATTERY INDICATOR; MASTER only; 1=low.
WRITE : TIMER ENABLE; 1=enable.

5.7 RAM

The 1LF3 RAM consists of 4096 bits of static hard configured memory arranged as 1024 4-bit nibbles.

5.8 Timer

The timer is a 6 nibble read/writeable binary counter located at addresses 2EnF8-2EnFD. The timer is decremented 512 times per second by the *CLK signal. The *CLK signal is derived from the MASTER's 32768 Hz quartz crystal oscillator and is also used to time display output signals. The MASTER chip requires a 32768 Hz crystal connected between OSCA and OSCB.

IMPORTANT: Because the timer decrement is asynchronous with the timing on the HP-71 bus, it is possible that a decrement will occur between the reading or writing of nibbles to the timer. This can cause erroneous values to be read from or written to the timer. It can be avoided by reading the timer twice within 1/512 second and verifying that both values are the same. If the values are not the same, immediately read the timer a third time. To write the timer, repeatedly read the least significant nibble until it decrements, then immediately write the timer value.

If the timer's most significant bit is a 1 and it is enabled by BIT3 of the display-timer control nibble the chip will respond to a service POLL by pulling BUS[0] low and it will wake up the CPU during shutdown by pulling *CD low. After waking the CPU the chip will not wake the CPU again until either the MSB of the timer or BIT3 of the control nibble has been cleared and set again.

The HP-71 operating system uses the timers to implement the real time clock system. For information on the clock system see the HP-71 software IDS, Volume 1.

5.9 Low Battery Indicator

The low battery indicator senses 2 power supply levels: low battery and very low battery. The 2 bits LBI and VLBI are read from BIT2 and BIT3 of the display-timer control nibble and are valid only on the MASTER. After each read of the control nibble a new sample of the supply voltage begins. This sample requires 100us. After the sample is complete the LBI and VLBI bits are updated. If the control nibble is read before the sample is complete the old low battery values are read and a new sample begins.

The HP-71 operating system samples the LBI bit once each minute. If it is true, the BAT annunciator will be lit. The operating system ignores the VLBI bit.

```
+----------------------------------------------+-------------------+
|                                              |                   |
|   ROM CHIP                                   |   CHAPTER  6      |
|                                              |                   |
+----------------------------------------------+-------------------+
```

The 1LG7 ROM is designed to support the 1LF2 CPU and future processors that operated on the HP-71 bus. The ROM core consists of 128K bits of memory arranged as 32,768 four-bit nibbles.

6.1 Pin Designations

The pins of the 1LG7 ROM chip are as follows:

PIN	FUNCTION
VDD	Power Supply.
GND	System Ground.
BUS[0:3]	System bus.
*STR	*STROBE
*CD	*COMMAND-DATA
OD	OUTPUT DISABLE - when driven high, the ROM tri-states the BUS[0:3]; OD is passively pulled low on-chip by an internal resister.
DIN	Daisy chain input.
DOUT	Daisy chain output.

In the HP-71 system, the OD pins of the 4 system ROMs are tied together. This signal is available at PORT1, and with special hardware, at PORT3 and the HP-IL port. By pulling the OD line high, all 4 system ROMs are effectively removed from the bus.

6.2 Bus Commands

The 1LG7 ROM can be either hard or soft configured and has no service request capability. The ROM responds to the bus commands described in section 2.1.2 with the following exceptions:

1	ID	If hard configured : same as NOP.
4	PC WRITE	The ROM increments its local program counter once each data strobe. No write is performed.
5	DP WRITE	The ROM increments its local data pointer once each data strobe. No write is performed.
8	CONFIGURE	If hard configured : same as NOP.
9	UNCONFIGURE	If hard configured : same as NOP.
A	POLL	Same as NOP.
C	BUSCC	Same as NOP.
E	SHUTDOWN	Same as NOP.
F	RESET	If hard configured : same as NOP.

The ROM's ID code is hard programmed (if soft configured). Generally, the ID code will be either 0010A, for one ROM of a multiple ROM set, or 8010A for an individual ROM or the last ROM of a multiple ROM set.

6.3 Addressing

The 32K nibbles of ROM require 15 bits of address space, leaving 5 bits of configuration address. The chip is selected when the upper 5 bits of the PC or DP (whichever is active) match the 5 bits stored in its configuration register and its configuration flag is set. The chip uses the remaining 15 bits to address its memory.

The ROM chip is manufactured in both soft and hard configured options (see section 2.2). In the hard configured option the 5 bits of the configuration register as well as the configuration flag are mask programmed to configure the ROM chip to a fixed address. The HP-71 operating system is stored in 4 mainframe ROMs configured as follows:

		ROM0	ROM1	ROM2	ROM3
Starting addr	=	00000	08000	10000	18000
Ending addr	=	07FFF	0FFFF	17FFF	1FFFF

Some plug-in ROMs are soft configured. In the soft configured option, the configuration register latchs the configuration address under software control as described in section 2.2.1.

RAM CHIP	CHAPTER 7

The 1LG8 RAM is designed to support the 1LF2 CPU and future processors that operate on the HP-71 bus. The RAM core consists of 8K bits of static memory arranged as 2048 four-bit nibbles.

7.1 Pin Designations

The pins of the 1LG8 RAM chip are identical to the pins of the 1LG7 ROM chip (see section 6.1) with the following addition:

ID   If tied high on the hybrid PC board (last chip) the most significant bit of the most significant nibble of the 5-nibble ID code will be set to a 1.

7.2 Bus Commands

The 1LG8 RAM is soft configured and has no service request capability. It responds to the bus commands as described in section 2.1.2 with the following exceptions:

A POLL   Same as NOP.

C BUSCC   Same as NOP.

E SHUTDOWN   Same as NOP.

The RAM ID code is n000E, where n=0 if the ID pin is tied low and n=8 if the ID pin is tied high. The ID pin is tied high only on the last chip of the 4-chip hybrid.

7.3 Addressing

The 2K nibbles of RAM require 11 bits of address space, leaving 9 bits of configuration address. The chip is selected when the upper 9 bits of the PC or DP (whichever is active) match the 9 bits stored in its configuration register and its configuration flag is set. The chip then uses the remaining 11 bits to address its memory. As an example of addressing, if a 4-chip RAM hybrid has been configured contiguously starting at 30000 hex, the following would apply:

		RAM0	RAM1	RAM2	RAM3
Starting addr	=	30000	30800	31000	31800
Ending addr	=	307FF	30FFF	317FF	31FFF

SYSTEM DIAGRAMS | CHAPTER 8

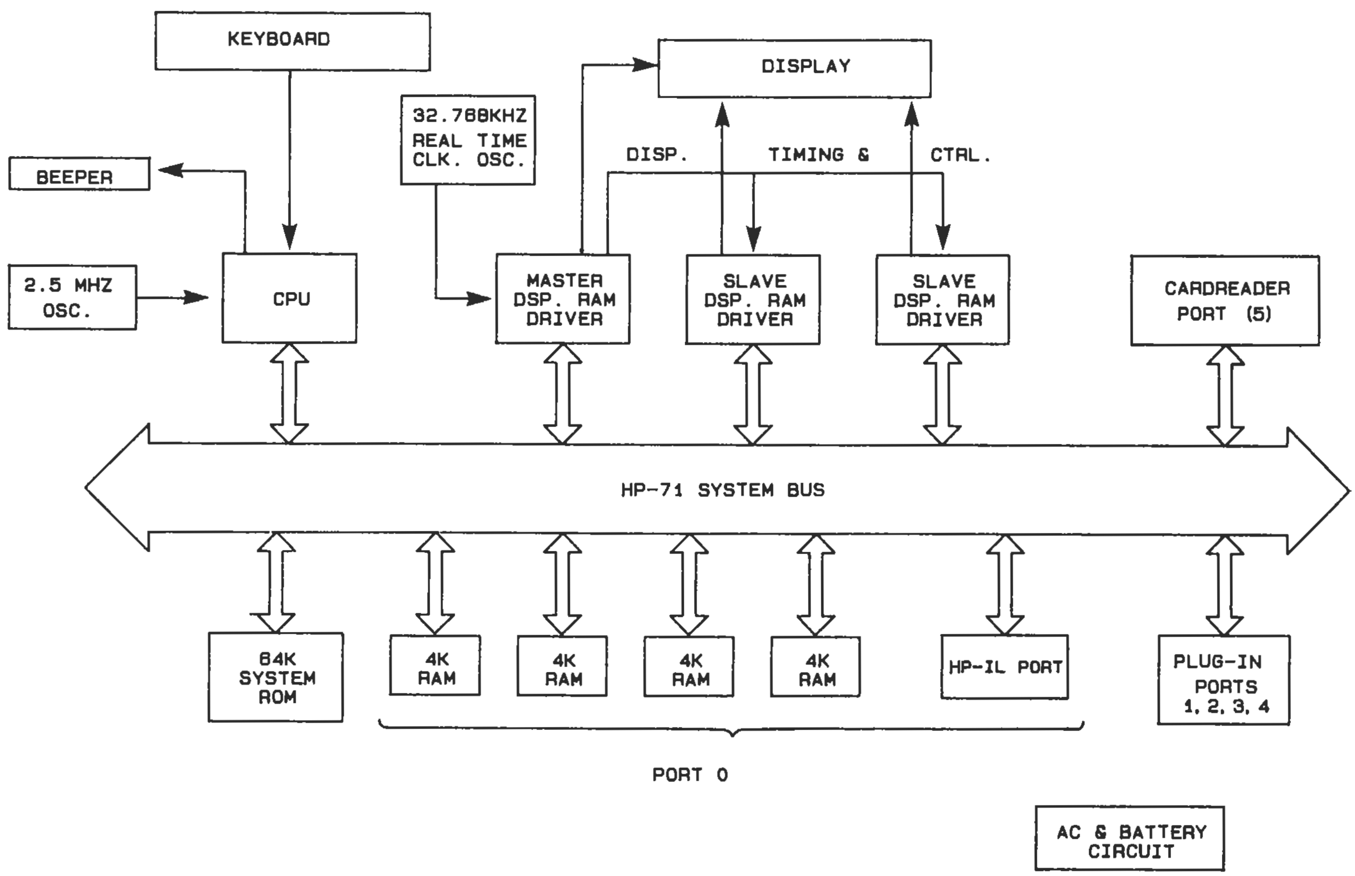


Figure 8-1. System Block Diagram

Table 8-1. Top-Case Assembly Replaceable Parts

INDEX NUMBER, FIGURE 7-1	HP PART NUMBER	DESCRIPTION	QUANTITY
C4,C5	0160-5789	CAPACITOR, 33 pF, 50V, 5%	2
C6,C7	0160-5790	CAPACITOR, 0.1 uF, 25V, 20%	2
C9,C10	0160-5787	CAPACITOR, 1000 pF, 5V, 20%	2
C11,C14	0160-5788	CAPACITOR, 220 pF, ±5%	2
L1	9140-0802	INDUCTOR, 180 uH, 5%	1
R20,R21	0699-1141	RESISTOR, 10K, 5%, 1/8W	2
Y1	0410-1381	CRYSTAL, quartz	1

Table 8-2. I/O Assembly Replaceable Parts

INDEX NUMBER, FIGURE 7-2	HP PART NUMBER	DESCRIPTION	QUANTITY
C1	0180-3351	CAPACITOR, 470 uF, 10V, 20%	1
C2	0180-3352	CAPACITOR, 330 uF, 16V, 20%	1
C8	0160-0576	CAPACITOR, 0.1 uF, 50V, 20%	1
C12	0160-3879	CAPACITOR, 0.01 uF, 20%	1
C13	0160-4441	CAPACITOR, 0.47 uF, 10%	1
CR2	1901-0999	DIODE, Schottky	1
CR3	1906-0069	DIODE, bridge, full-wave, 400V	1
CR7	1901-0704	RECTIFIER, silicon	4
L2	9140-0794	INDUCTOR, 56 mH, 10%	1
Q1	1854-0932	TRANSISTOR, NPN	1
Q2	1854-0973	TRANSISTOR,	1
R22	0683-3915	RESISTOR, 390 ohms, 5%, 0.25W	1
R23,R24	0683-1035	RESISTOR, 10 K-ohms, 5%, 0.25W	2
VR7	1902-1390	DIODE, Zener	1

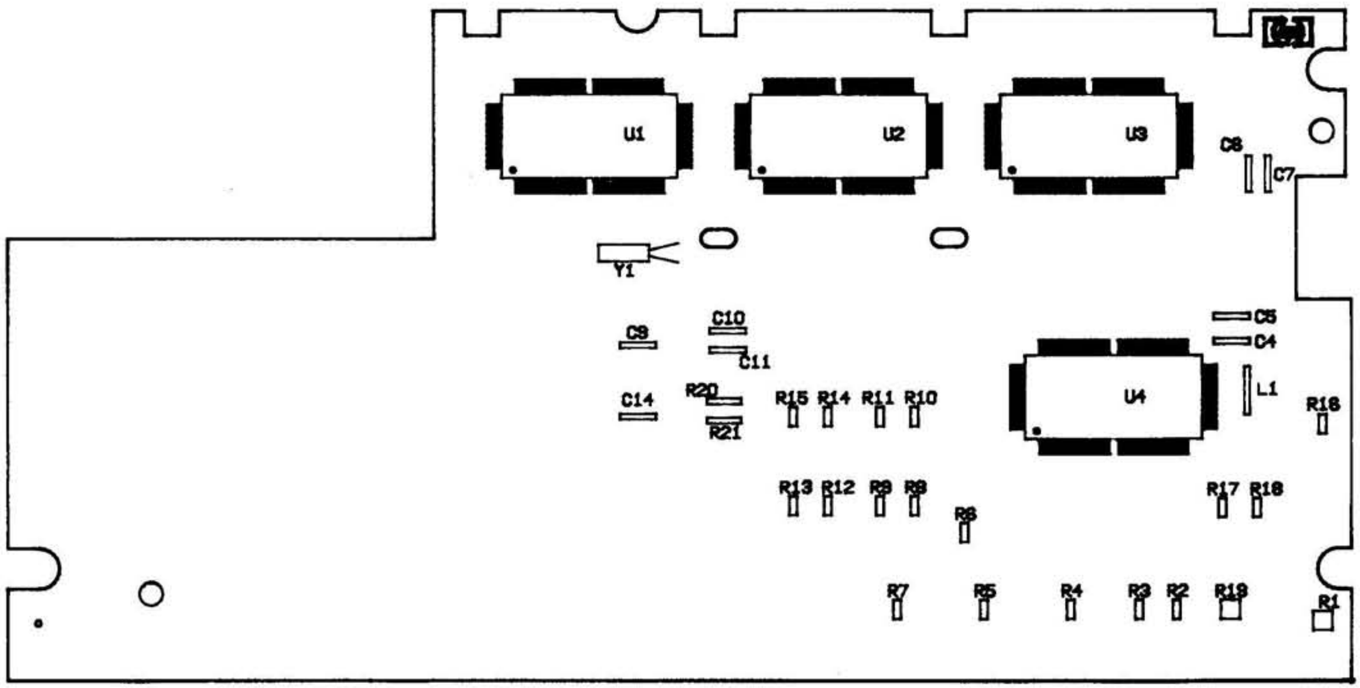


Figure 8-2. Top-case Assembly Component Location Diagram

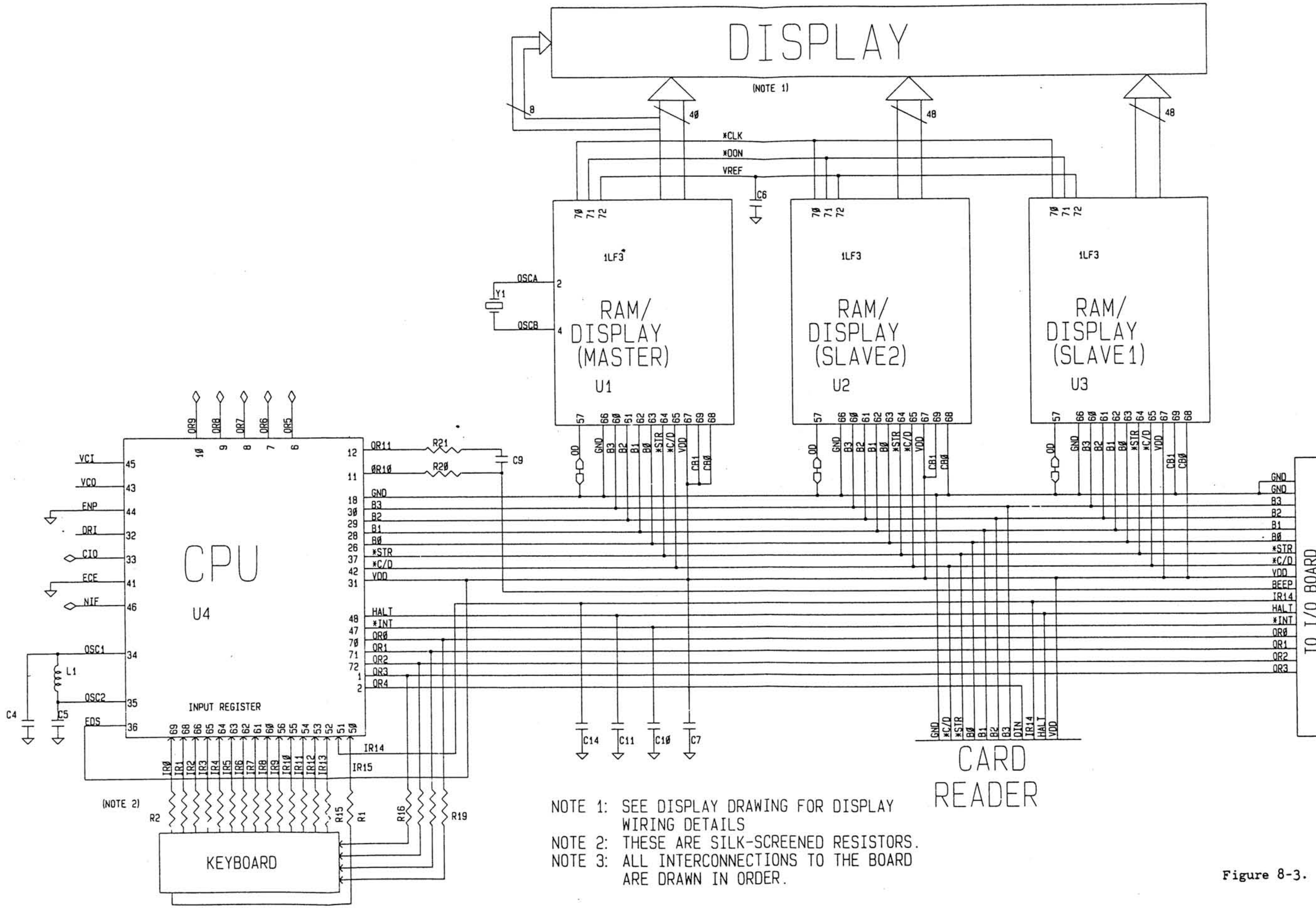

DISPLAY
(NOTE 1)
*CLK
*DON
VREF
RAM/ DISPLAY (MASTER)
U1
RAM/ DISPLAY (SLAVE2)
U2
RAM/ DISPLAY (SLAVE1)
U3
1LF3
OSCA
OSCB
Y1
CPU
U4
INPUT REGISTER
VCI
VCO
ENP
DRI
CIO
ECE
NIF
OSC1
OSC2
EOS
L1
HALT
*INT
KEYBOARD
(NOTE 2)
CARD READER
TO I/O BOARD
GND
B3
B2
B1
BØ
*STR
*C/D
VDD
BEEP
IR14
IR15
ORØ
OR1
OR2
OR3
OR4
NOTE 1: SEE DISPLAY DRAWING FOR DISPLAY WIRING DETAILS
NOTE 2: THESE ARE SILK-SCREENED RESISTORS.
NOTE 3: ALL INTERCONNECTIONS TO THE BOARD ARE DRAWN IN ORDER.

Figure 8-3. T

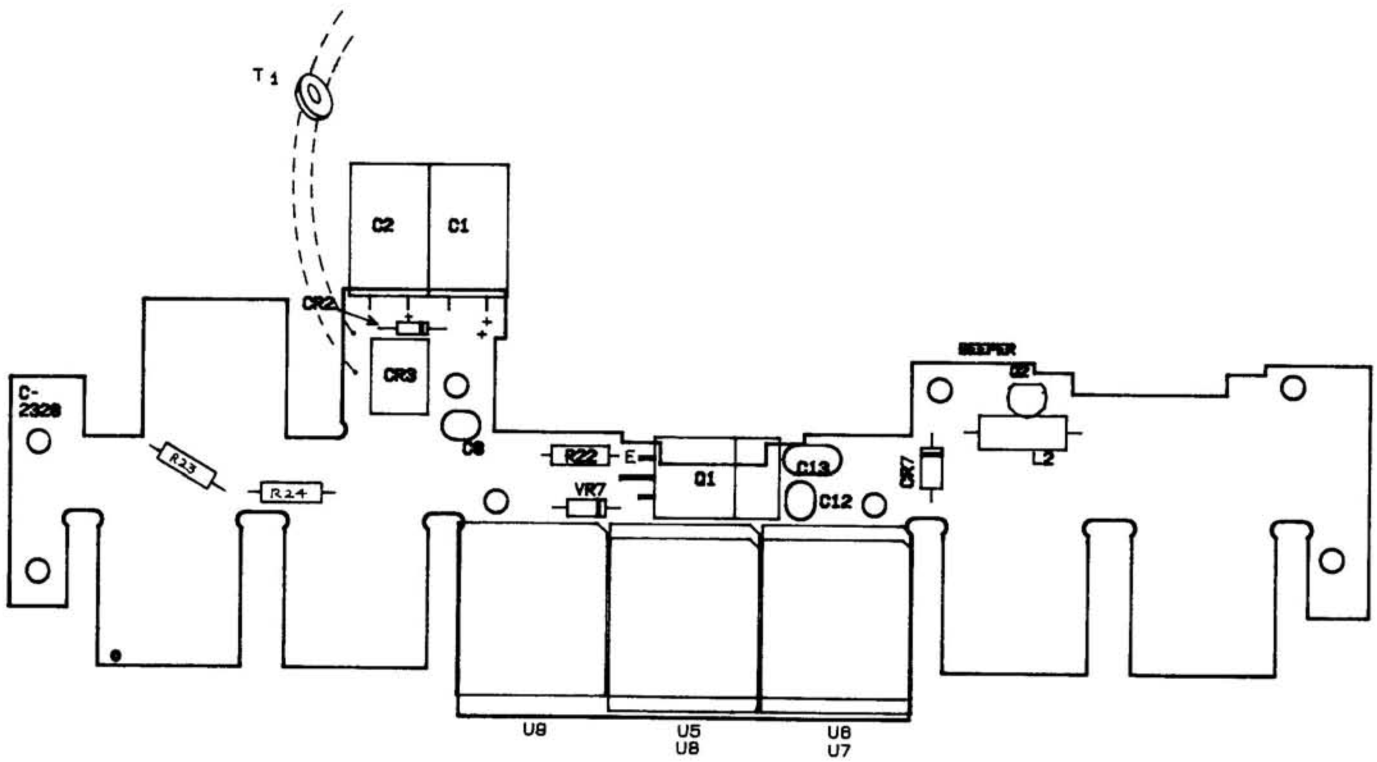
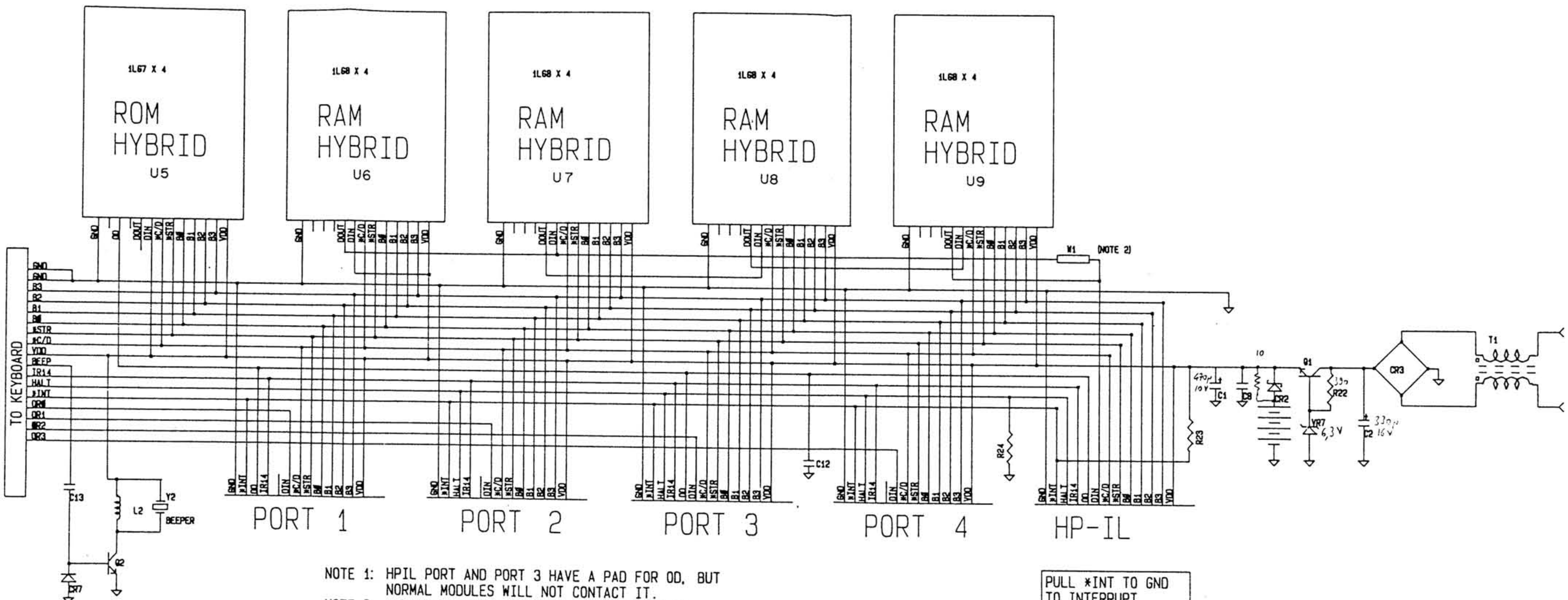


Figure 8-4. I/O PCA Component Location Diagram



NOTE 1: HPIL PORT AND PORT 3 HAVE A PAD FOR OD, BUT NORMAL MODULES WILL NOT CONTACT IT.

NOTE 2: ALL INTERCONNECTIONS ARE DRAWN IN ORDER.

Figure 8-5. I/O PCD Sc

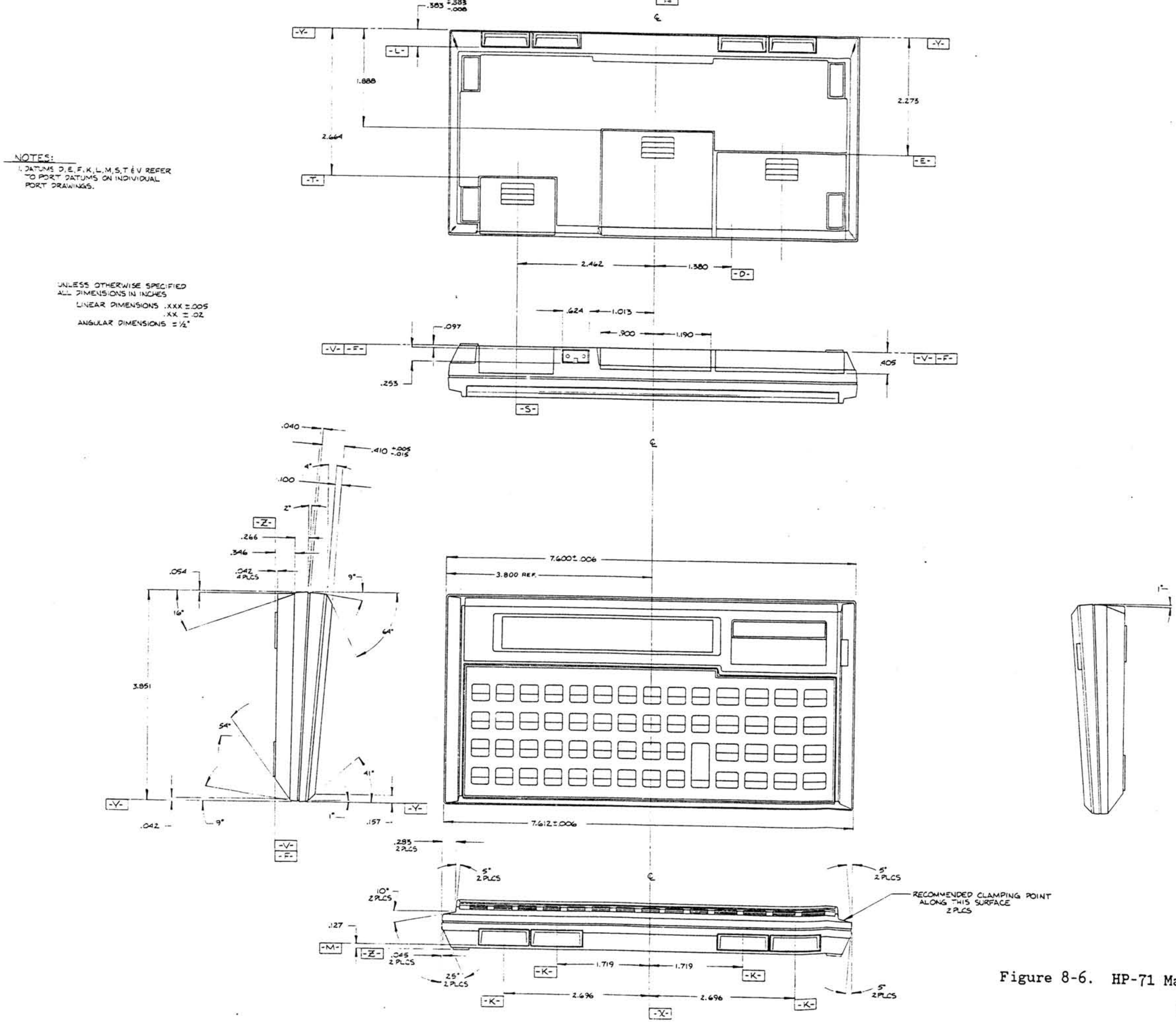


Figure 8-6. HP-71 Mainframe

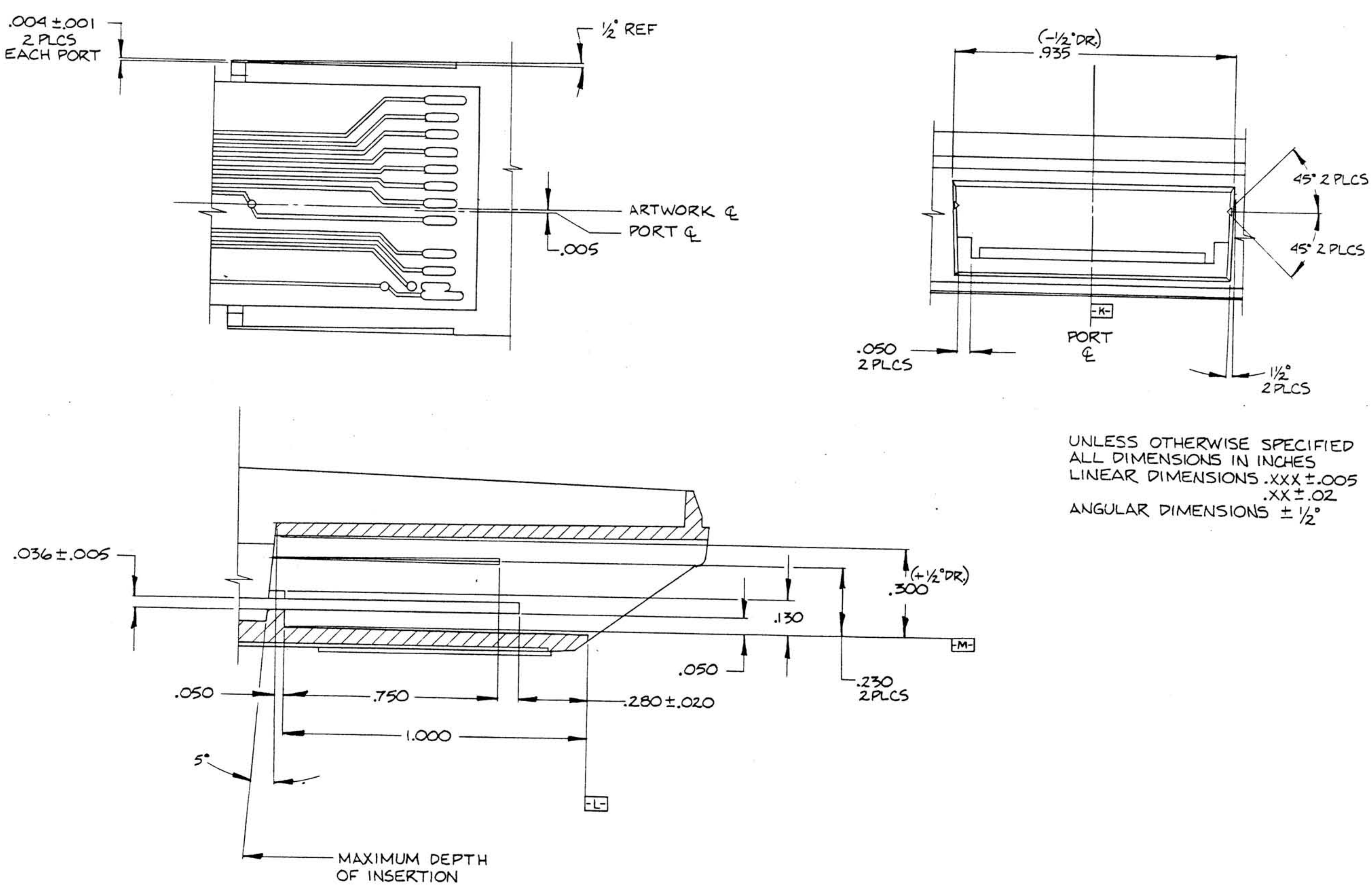


Figure 8-7. HP-71 Memory Module Port

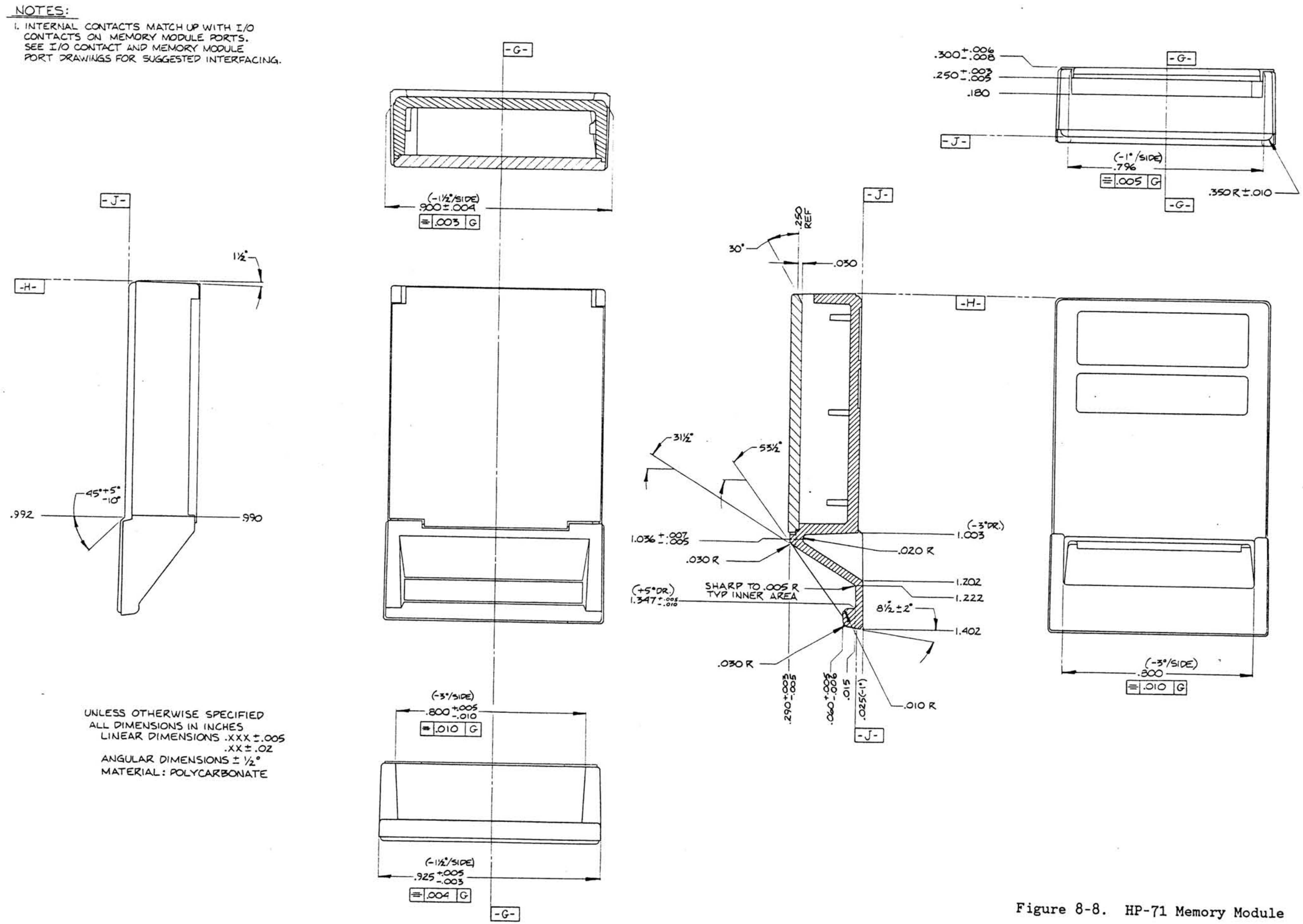
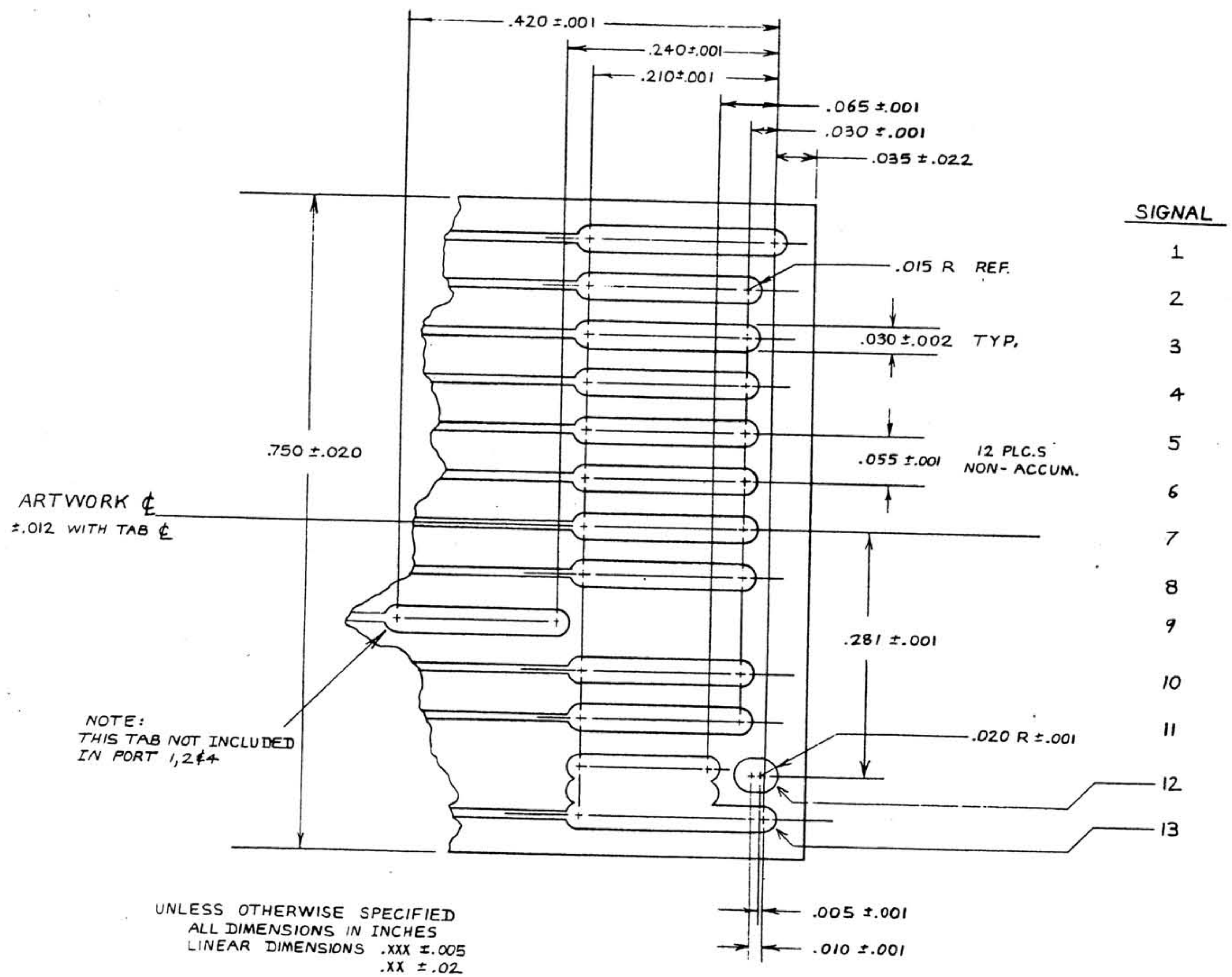


Figure 8-8. HP-71 Memory Module



		MEMORY				HPIL
		PORT 1	PORT 2	PORT 3	PORT 4	PORT
SIGNAL	1	VDD	VDD	VDD	VDD	VDD
	2	B3	B3	B3	B3	B3
	3	B2	B2	B2	B2	B2
	4	B1	B1	B1	B1	B1
	5	BØ	BØ	BØ	BØ	BØ
	6	*STR	*STR	*STR	*STR	*STR
	7	*C/D	*C/D	*C/D	*C/D	*C/D
	8	DIN/ORØ	DIN/OR1	DIN/OR2	DIN/OR3	DIN
	9	—	—	OD	—	OD
	10	IR 14	IR 14	IR 14	IR 14	IR 14
	11	O.D.	HALT	HALT	HALT	HALT
	12	*INT	*INT	*INT	*INT	*INT
	13	GND	GND	GND	GND	GND

RECOMMENDED PLATING

.5 mils COPPER CLAD
.8 mils PLATED COPPER MIN.
.2 mils NICKEL MIN.
.03 mils GOLD MIN.

Figure 8-9. HP-71 I/O Contact

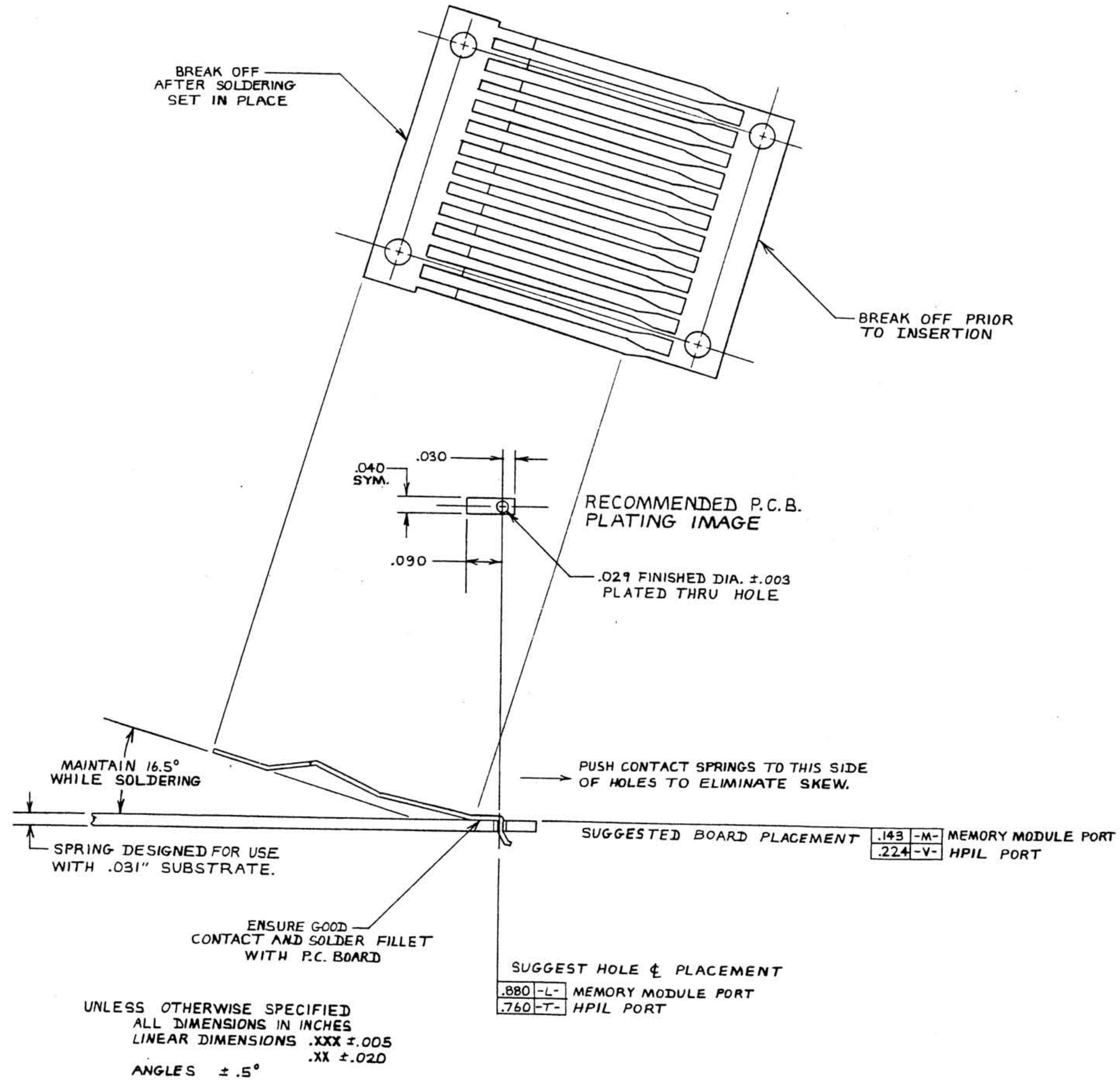


Figure 8-10. HP-71 Contact Spring Attachment Diagrams

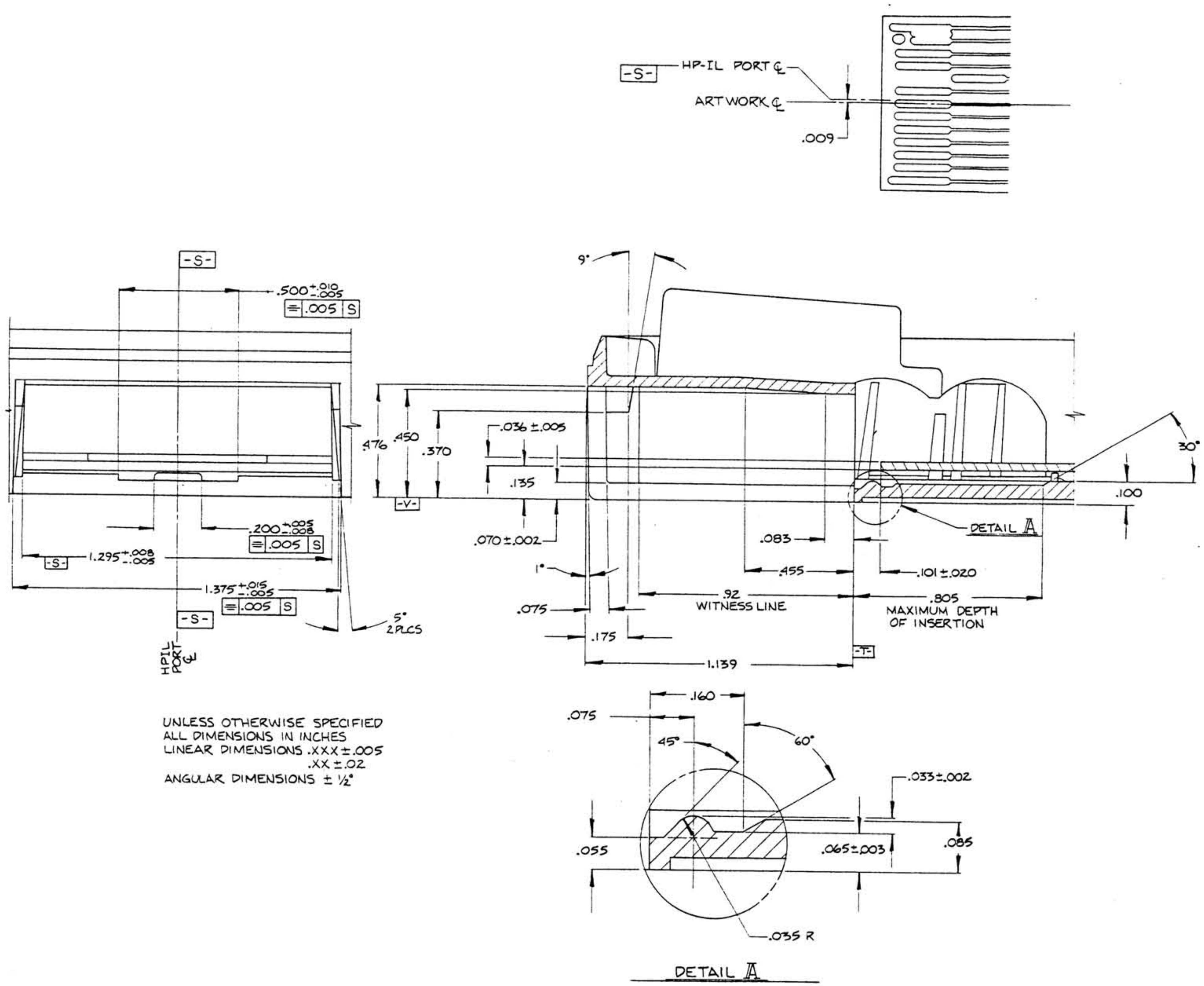


Figure 8-11. HP-71 HP-IL Module Port

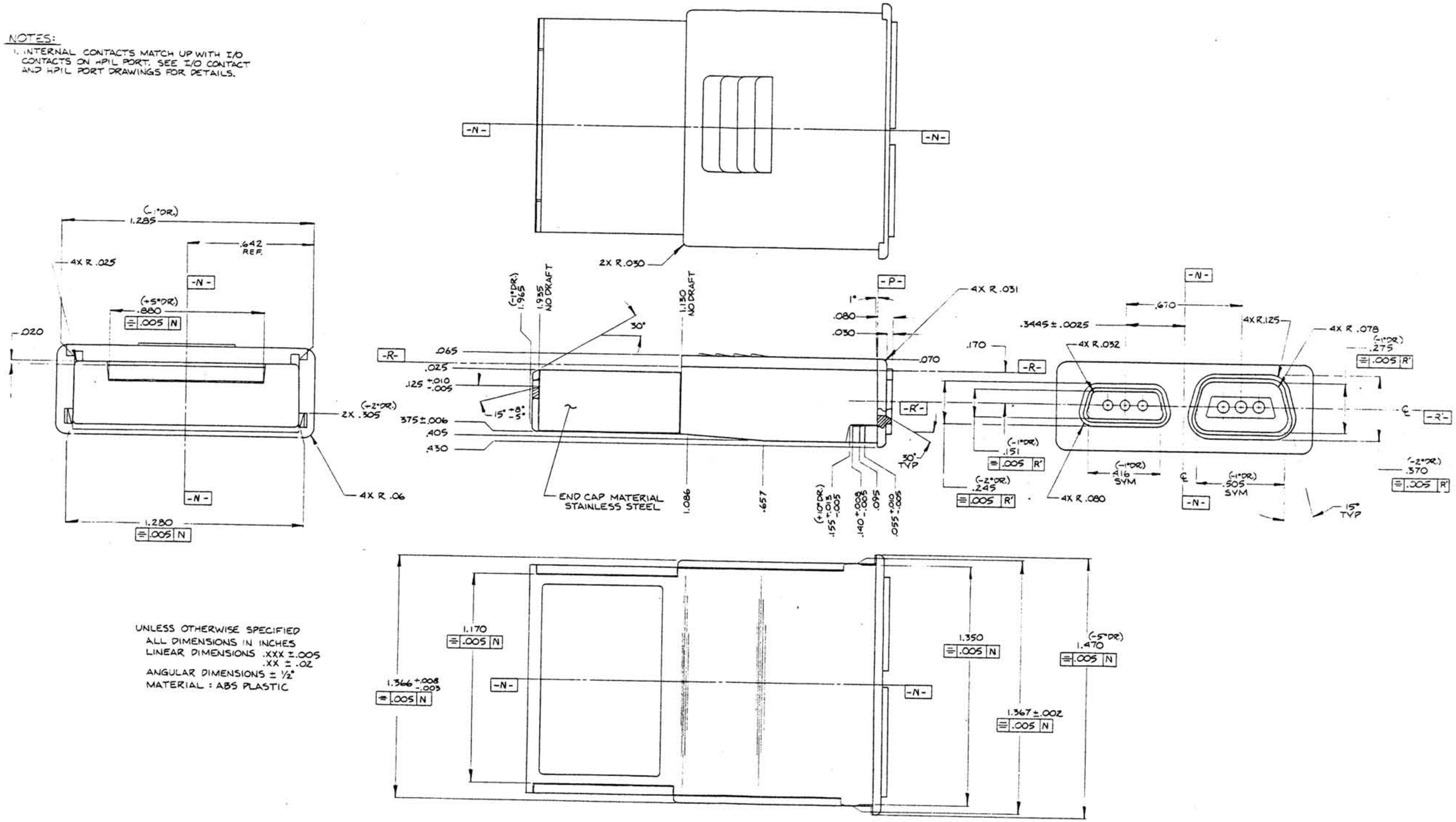


Figure 8-12. HP-71 HP-IL Module

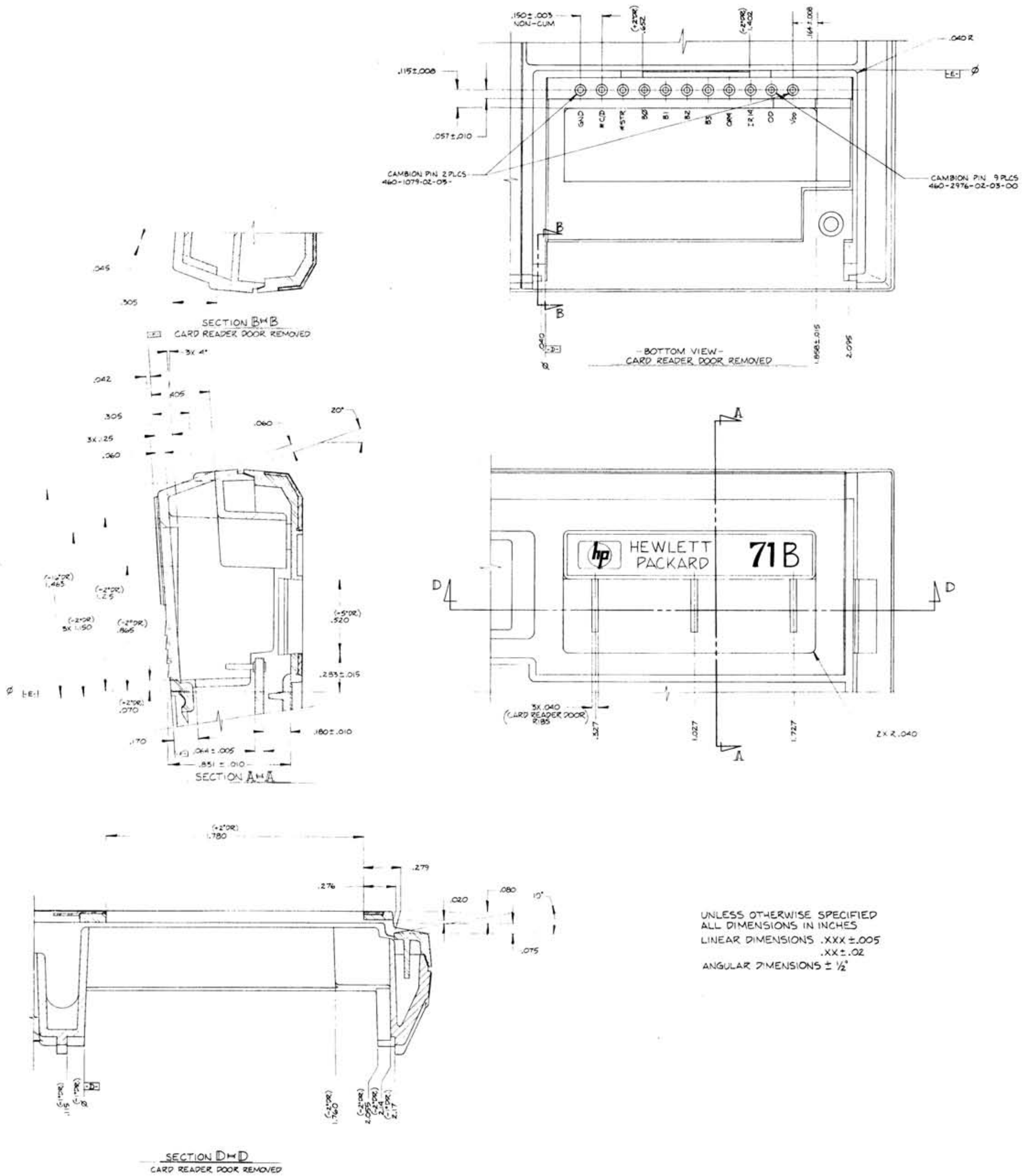


Figure 8-13. HP-71 Card Reader Port

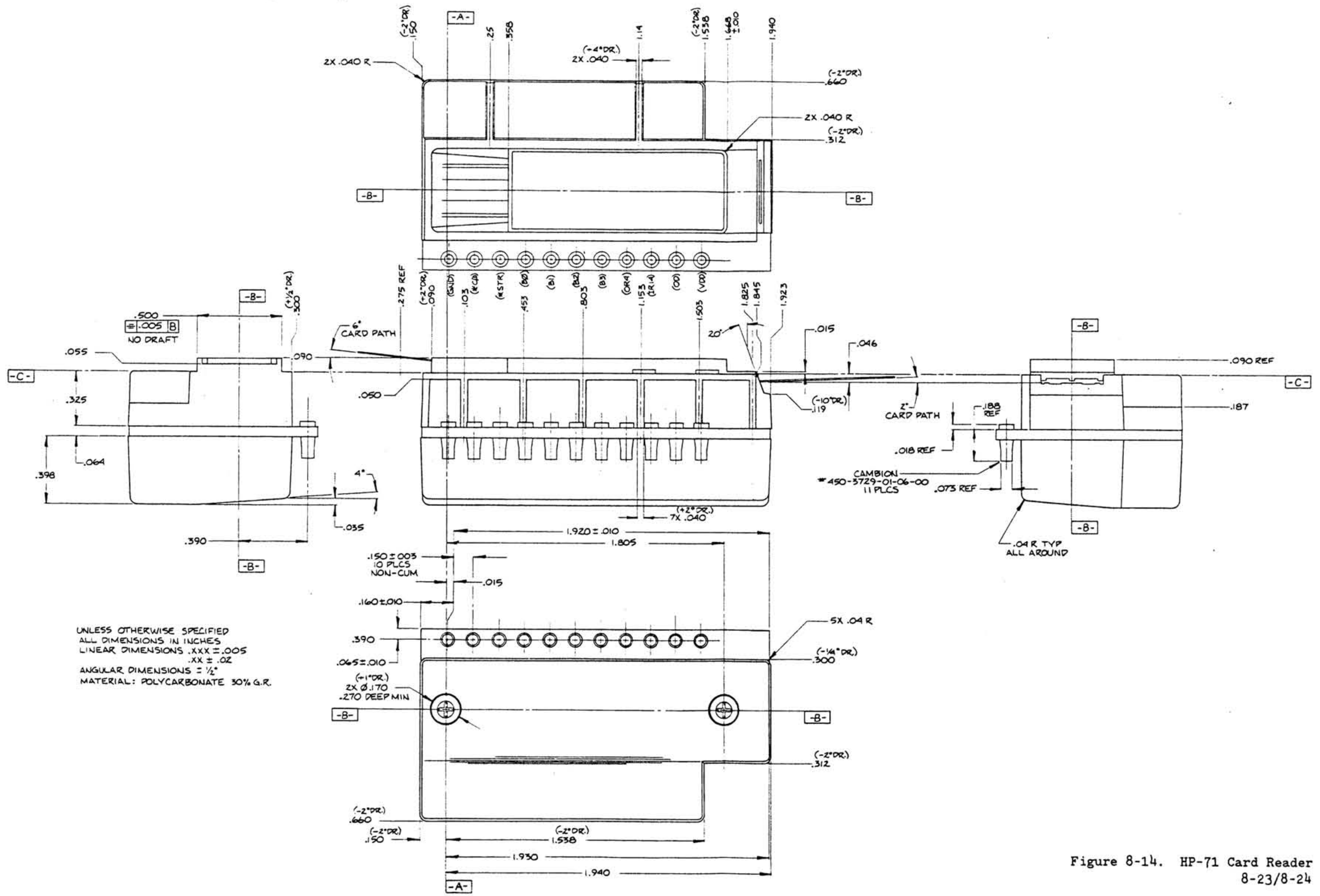


Figure 8-14. HP-71 Card Reader

SYSTEM ELECTRICAL SPECIFICATION	CHAPTER 9

IC ABSOLUTE MAXIMUM RATINGS:

Supply Voltage Vdd .... Vss + 7.5V

Maximum Voltage at any Input or Output ..... Vdd + 0.3V

Minimum Voltage at any Input or Output ..... Vss - 0.3V

Operating Free-Air Temperature .... 0C to +45C

Storage Temperature .... -40C to +55C

Humidity .... 0 to 95% RH

TEST CONDITIONS: Test conditions are such that the following parameters are guaranteed for Vdd = 4.25V to 6.5V and Temperature = 0C to 45C unless a different supply voltage or temperature is noted.

SYM	PARAMETER	MIN	TYP	MAX	UNIT	COMMENTS
Vss	Ground	0.0	0.0	0.0	V	
Vdd	Supply voltage	4.25	-	6.5	V	
Iddop	Idd operating current	-	-	15.0	mA	Display on, CPU running, T0=1.5uS.
Iddls	Idd light sleep current	-	-	0.5	mA	Display on, CPU shutdown.
Iddds	Idd deep sleep current	-	-	50	uA	Display off CPU shutdown.

SYM	PARAMETER	MIN	TYP	MAX	UNIT	COMMENTS
POWER SUPPLY						
Vac	AC adapter input voltage	9.0	-	14.0	V	See Note 1.
Vreg	AC adapter voltage at Vdd	5.6	-	6.5	VDC	
Iac	AC adapter current drain	-	-	30	mADC	Over mainframe requirements.
Vbat	Battery voltage at Vdd	4.25	-	6.0	VDC	Unregulated.
Ibat	Battery current drain	-	-	100	mADC	Over mainframe requirements.
BUS PARAMETERS						
Vih	Input logic level '1'	Vdd-.65	-	-	V	
Vil	Input logic level '0'	-	-	Vss+.65	V	
Voh	Output logic level '1'	Vdd-.5	-	-	V	
Vol	Output logic level '0'	-	-	Vss+.5	V	
Cout	Output capacitance drive capability (surplus)					
	BUS0-3	50	-	-	pF	See Notes 2 & 3.
	*CD,*STR	50	-	-	pF	See Notes 2 & 3.
	DIN	10	-	-	pF	With no pulldown, See Note 4.
	DIN	50	-	-	pF	With Rpd = 50Kohm, See Note 4.

SYM	PARAMETER		MIN	TYP	MAX	UNIT	COMMENTS
Ioh	High level output source current capability						
		BUS0-3	1.0	-	-	mADC	See Note 3.
		*CD,*STR	1.0	-	-	mADC	See Note 3.
		DIN	100	-	-	uADC	
Iol	Low level output sink current capability						
		BUS0-3	1.0	-	-	mADC	See Note 3.
		*CD,*STR	1.0	-	-	mADC	See Note 3.
		DIN	2	-	-	uADC	
Cin	Input capacitance loading						
		BUS0-3	-	-	300	pF	See Note 2.
		*CD,*STR	-	-	250	pF	See Note 2.
		*INT	-	-	1200	pF	
		IR14	-	-	250	pF	
		HALT	-	-	250	pF	
		OD	-	-	.012	uF	
Iih	High level input current loading						
		BUS0-3	-	-	50	uADC	Internal pulldown
		*CD	-	0.0	-	uADC	Internal pullup
		*STR	-	0.0	-	uADC	Internal pullup
		*INT	-	0.0	-	uADC	10K pullup
		IR14	-	-	50	uADC	Internal pulldown
		HALT	-	-	1.0	mADC	10K pulldown
		OD	-	-	50	uADC	Internal pulldown

SYM	PARAMETER	MIN	TYP	MAX	UNIT	COMMENTS
Iil	Low level input current loading					
	BUS0-3	-	0.0	-	uADC	Internal pulldown
	*CD	-	-	50	uADC	Internal pullup
	*STR	-	-	50	uADC	Internal pullup
	*INT	-	-	1.0	mADC	10K pullup
	IR14	-	0.0	-	uADC	Internal pulldown
	HALT	-	0.0	-	uADC	10K pulldown
	OD	-	0.0	-	uADC	Internal pulldown

BUS TIMING PARAMETERS

T0	*STR cycle time	1.0	-	-	us	Current HP-71's operate at 600KHz to 650KHz, future HP-71's will run at up to 1MHz.
Tpwl	*STR low	0.5	-	-	us	
Tpwh	*STR high	0.5	-	-	us	
Tdwc	Data-in valid to *STR high	200	-	-	ns	Command cycle
Tdwd	Data-in valid to *STR high	100	-	-	ns	Write cycle
Tdh	*STR high to data-in invalid	100	-	-	ns	
Tacc	*STR low to data-out valid	-	-	200	ns	BUS precharged low
Toh	*STR high to data-out tristated	20	-	100	ns	

SYM	PARAMETER	MIN	TYP	MAX	UNIT	COMMENTS
Tc1	*CD low to *STR low	30	-	-	ns	
Tc2	*STR high to *CD high	50	-	-	ns	
Tc3	*CD high to *STR low	100	-	-	ns	
Tr	Rise time					
	*STR	-	-	100	nS	
	*CD	-	-	100	nS	
	BUS0-3	-	-	100	nS	
Tf	Fall time					
	*STR	-	-	100	nS	
	*CD	-	-	100	nS	
	BUS0-3	-	-	100	nS	

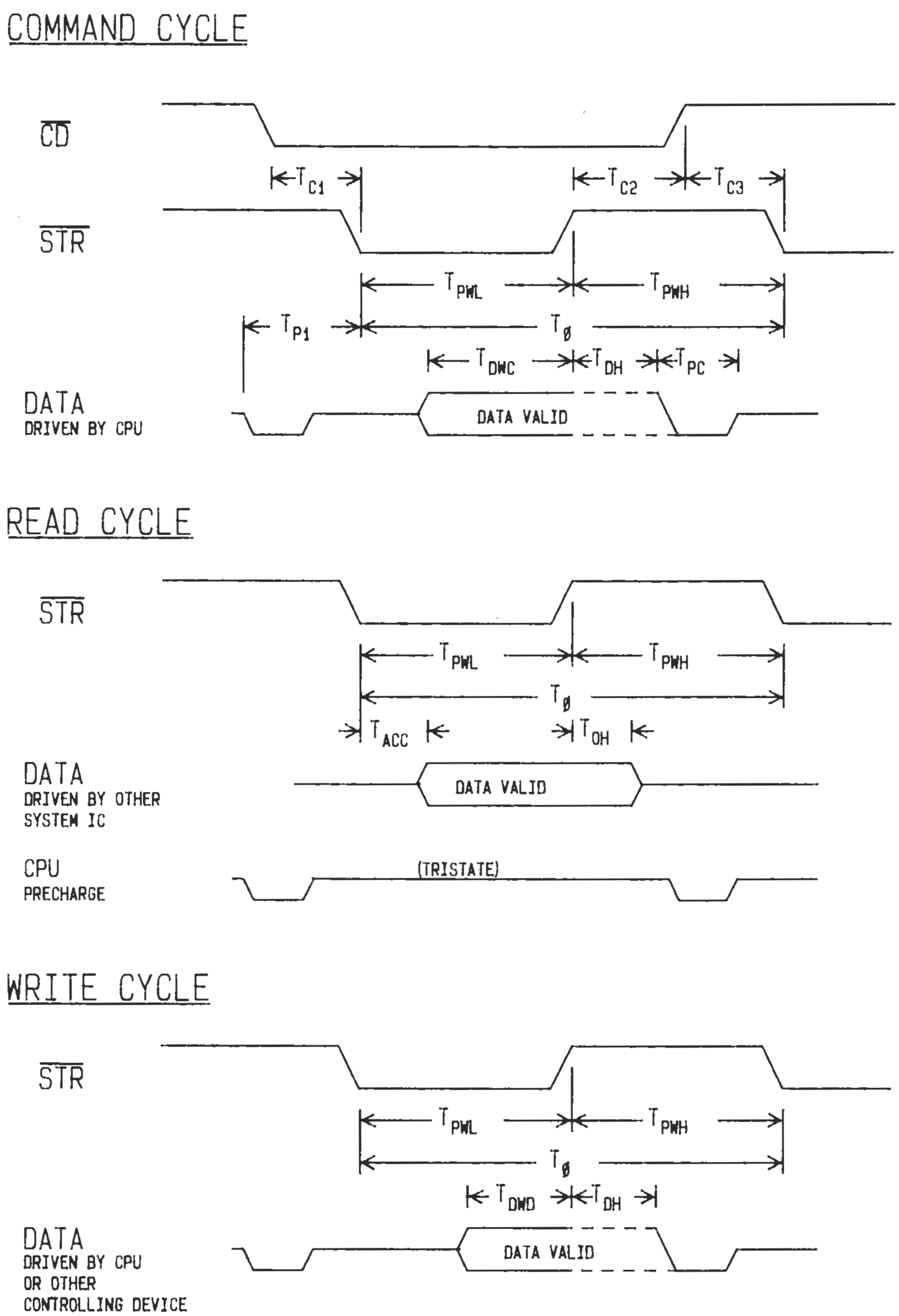


Figure 9-1. Bus Timing Relationships

SYM	PARAMETER	MIN	TYP	MAX	UNIT	COMMENTS
Fclk	Display *CLK frequency	-	512	-	Hz	+- 40 PPM Real time base.
LOW BATTERY INDICATION						
Vlbi	Low battery trip point	4.3	-	4.5	V	Low battery bit high when Vdd < Vlbi.
Vdta	Vlbi - Very low battery trip point	.08	-	.12	V	Very low battery bit high when Vdd < Vlbi-Vdta.

NOTES :

1) The values specified for Vac limit the peak voltage that may be applied between the 2 AC adapter input pins. This voltage may be either an AC peak voltage or a DC voltage. If a voltage less than 9.0V is applied the unit's batteries may be discharged. This minimum value for Vac (9.0V) can be reduced to 7.5V if the batteries are removed from the unit.

2) The values specified under Cout and Cin for BUS0-3, *CD, and *STR allow for 100pF loading by plug-in modules. This can be broken down as:

   a) RAM or ROM modules = 20pF x 4
   b) HP-IL module = 12pF
   c) Card reader module = 8pF

   A HP-71B with no plug-in modules could drive 100pF more than specified and would load these lines 100pF less than specified.

3) Reduce Cout (output capacitance drive capability) by 25 pF/mA of the larger of Ioh and Iol (high level source and low level sink current capability) required of BUS0-3, *CD, and/or *STR. Under no condition should the minimum current capability specified for Ioh and Iol be exceeded.

4) The DIN lines of the I/O ports are driven by the CPU output register (OR). The OR lines are also used to form the keyboard matrix and therefore excessive loading of a DIN line will prevent proper keyboard operation. The maximum capacitive loading of a DIN line may be increased to 50pF if a pulldown resistor, Rpd, of 50 Kohm (+-10%) is tied from DIN to ground. For most of the time when the HP-71 is turned on all lines of the OR are high and thus Rpd will draw current. When the HP-71 is shut off the operating system sets OR0 high and all other OR lines low. OR0 drives the DIN line of Port 1.

HEWLETT PACKARD

Portable Computer Division
1000 N.E. Circle Blvd., Corvallis, OR 97330, U.S.A.

European Headquarters
150, Route du Nant-D'Avril
P.O. Box, CH-1217 Meyrin 2
Geneva-Switzerland

HP-United Kingdom
(Pinewood)
GB-Nine Mile Ride, Wokingham
Berkshire RG11 3LL

00071-90071 English

Printed in U.S.A. 9/84