Part 3

Hardware Specifications

Introduction to the Hardware Specifications

The purpose of this section of the *Technical Reference Manual* is to provide enough information for a developer to test a system configuration using the HP-94 and accessories in typical usage. There are four major topics:

- Electrical Specifications
  This provides voltage and current levels and specific integrated circuit (IC) information for some of the system ICs.
- Mechanical Specifications
  This includes HP-94 dimensions and information about connector types and pin assignments.
- Environmental Specifications
  This provides temperature, humidity, and other environmental information about the HP-94 operating environment.
- Accessory Specifications
  This discusses the principal accessories currently available for the HP-94.

In addition, for reference by developers, data sheets are provided for four of the ICs used in the machine.

Disclaimer

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

The information contained in this document is subject to change without notice.

System Block Diagram

On the next page is a block diagram of the HP-94 hardware.

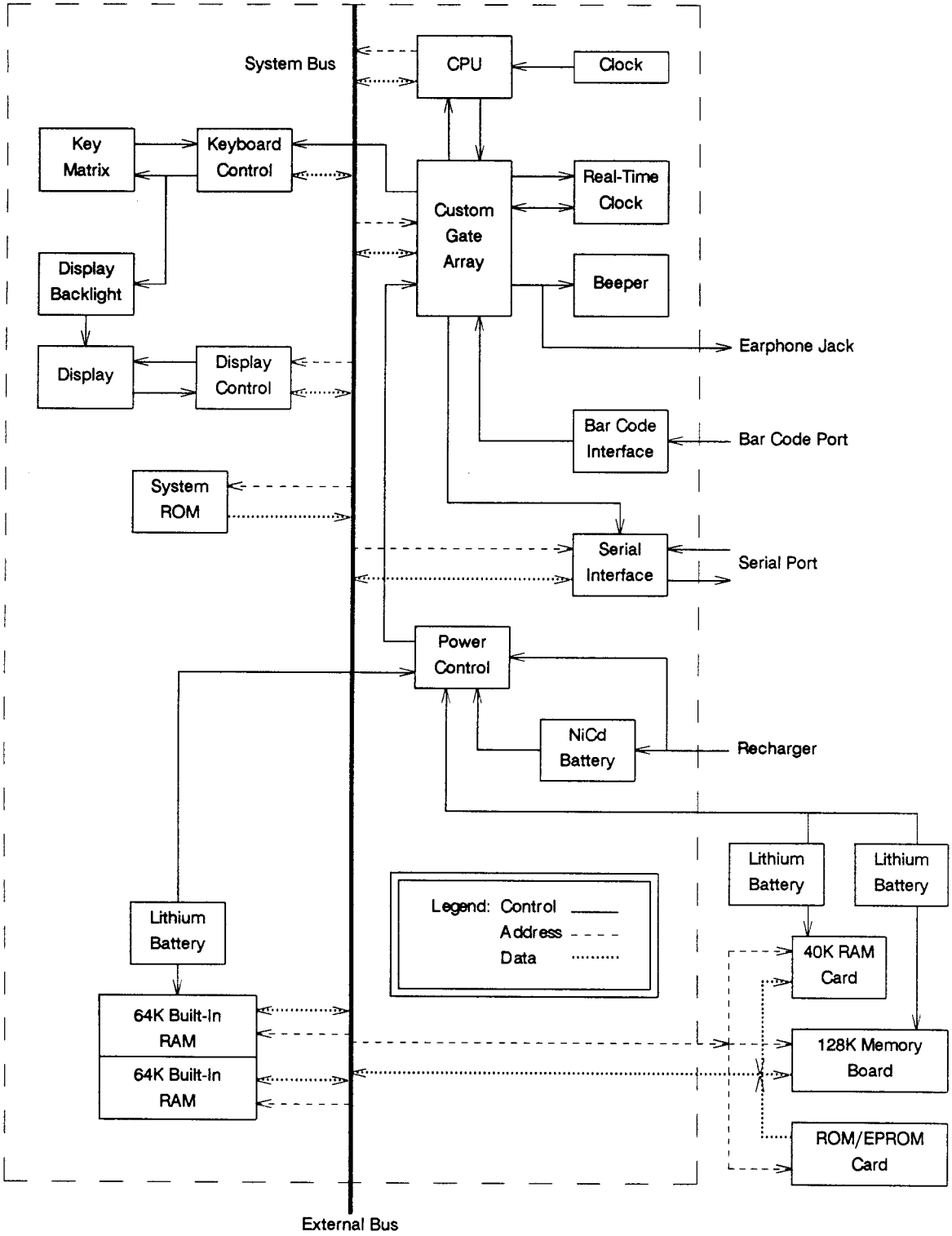


Figure 1. HP-94 Hardware Block Diagram

1

Electrical Specifications

1

Electrical Specifications

This section provides the basic electrical specifications for the HP-94. Specific bus timing information is not provided. This information is available in the manufacturer's specifications for the individual components. The principal ICs used in the HP-94 are shown below.

Table 1-1. Principal Integrated Circuits

IC	Manufacturer	Part Number
Microprocessor *	NEC	μPD70108 (V20)
RAM	Toshiba	TC5565FL-15L
EPROM	Toshiba	TC57256D-20
LCD Column Driver *	Hitachi	HD61102A
LCD Row Driver	Hitachi	HD61103A
UART *	OKI	MSM82C51A
Real-Time Clock *	Epson	RTC-58321
Custom Gate Array	Hitachi	61L224

* Refer to the data sheets for these devices.

Specific questions relating to the use of these ICs or their specifications should be directed to the IC manufacturer.

Table 1-2. Electrical Specifications

Parameter	Symbol	Min	Typical	Max	Units	Comments
Operating Voltage	V_{cc}	4.50	4.80	6.00	Vdc	Varies as batteries vary
Operating Current	I_{cc}		60 35	90 50	mA mA	Running Waiting for a key
Operating Frequency	F_{op}		3.6864		MHz	*
Display Backlight Current	I_{el}		20		mA	When backlight on
Standby Current	I_{sb}		30	100	µA	For HP-94F (256K RAM): T=25 °C
Battery Pack Capacity			900		mAh	HP 82430A Rechargeable NiCd Battery Pack
Low Battery Detect Level — NiCd	V_{llNi}	4.55 4.70	4.60 4.75	4.65 4.80	Vdc Vdc	Discharging (voltage decreasing) Charging (voltage increasing)
Low Battery Detect Level — Lithium	V_{llLi}	2.65	2.70	2.75	Vdc	
Reset Detect Level	V_{rst}	4.35	4.40	4.45	Vdc	
V_{cc} Cutoff Level	V_{cu}	3.80		4.20	Vdc	
Serial Port Source Current	I_{rs}	250			mA	For $V_{rs}=V_{cc}$-0.1 V
Serial Port Maximum Input Voltage	V_{inm}	-15		+15	Vdc	†
Serial Port Input Logic Levels	V_{ih} V_{il}	0.8 V_{cc}		 0.15 V_{cc}	Vdc Vdc	†
Serial Port Output Logic Levels	V_{ol1} V_{ol0}	V_{cc}-0.2 0		V_{cc} 0.2	Vdc Vdc	‡
Serial Port Input Current	I_{inm}			±10 ±3.5	µA mA	$V_{in}=V_{cc}$ or GND $V_{in}=\pm 15$ V
HP 82470A Level Converter Current	I_{lc}		25		mA	When active with a std. RS-232-C load at 9600 baud and $V_{cc}=6.0$ V
Bar Code Port Source Current	I_{bcr}	175			mA	For $V_{bcr}=V_{cc}$-0.1 V
Bar Code Port Input Logic Levels (V_{Obcr})	V_{ih} V_{il}	0.75 V_{cc} 0		V_{cc} 0.15 V_{cc}	Vdc Vdc	CMOS 40H004 inverting buffer
Bar Code Port Input Current	I_{inm}			+1 -6	µA mA	Input logic 1 § Input logic 0 §

* Voltage and timing specifications associated with the external bus and memory port are established by the CPU voltage and timing specifications. Please refer to the NEC µPD70108 data sheet.

† The HP-94 has a special input protection circuit using a 4.7 volt zener diode clamp that keeps the input voltage less than 4.7 volts and greater than -0.6 volts. Input signals between 0 and V_{cc} will not be modified.

‡ The output drivers are CMOS 40H004 inverting buffers that drive only between GND and V_{cc} voltage levels.

§ The bar code port input (V_{Obcr}) drives a 40H004 inverting buffer with a 1 kΩ pullup resistor.

2

Mechanical Specifications

Contents

Chapter 2 Mechanical Specifications

2

Mechanical Specifications

This chapter describes mechanical specifications for the HP-94 and its connectors.

Physical Specifications

Below are the physical specifications for the HP-94.

Table 2-1. Physical Specifications

Parameter	Value	Units	Comments
Height	16.0	cm	6.3 in
Width	16.5	cm	6.5 in
Thickness	3.7	cm	1.4 in
Weight	686-745 *	g	1.5-1.6 lb

* The weight varies depending on the memory configuration. Minimum is for the HP-94D, and maximum is for the HP-94E with an HP 82412A ROM/EPROM Card containing three 27C256 EPROMs.

Serial Port Connector Specifications

The serial port connector is a 15-pin D-type female connector. The connector's attachment bolts use 4-40 × 1/4" slotted-head screws; note that it is not necessary to secure cables to the machine using these bolts. The following tables provide serial port connector pin assignments and information about mating connectors for the serial port.

Table 2-2. Serial Port Connector Pin Assignments

Pin Number	Symbol	Signal Name
Housing	FG	Frame Ground
1	NC	Not Connected
2	TxD	Transmitted Data
3	RxD	Received Data
4	RTS	Request To Send
5	CTS	Clear To Send
6	DSR	Data Set Ready
7	SG	Signal Ground
8	DCD	Data Carrier Detect
9	V_{rs}	Switched V_{cc}
10	V_{rch} *	Alternate Recharger Input
11	GND	Recharger Ground Return
12-14	NC	Not Connected
15	DTR	Data Terminal Ready

* The specifications for Vrch to allow charging of the NiCd battery pack using this pin are the same values as the output voltage and current specifications of the HP 82431A recharger.

Table 2-3. Serial Port Mating Connectors

Manufacturer	Part Number
TRW	DAM 15P
Amphenol	17-20150-1
JAE	DAC 15P
ITT Cannon	DA-15P

Bar Code Port Connector Specifications

The bar code port uses a 6-pin, 240° circular DIN connector. Either a 5-pin or 6-pin mating connector can be used on the bar code reader since pin 6 is not connected at the bar code port. The following tables provide bar code port connector pin assignments and information about mating connectors for the bar code port.

Table 2-4. Bar Code Port Connector Pin Assignments

Pin Number	Symbol	Signal Name
1	V_{bcr}	Switched V_{cc}
2	V_{Obcr}	Input From Barcode Reader
3	GND *	Signal Ground
4-6	NC	Not Connected

* The connector housing is attached to signal ground.

Table 2-5. Bar Code Port Mating Connectors

Manufacturer	Part Number
Switchcraft	12BL5M
Switchcraft	12BL6M
ITT Cannon	46005F
ITT Cannon	46006
TRW	014-00016-5
TRW	014-00024-1
SMK	DIN45322

Memory Port Connector Specifications

The memory port connector is inside the back cover of the HP-94, and is where the 40K RAM card, ROM/EPROM card, and 128K memory board connect to the machine. It is a Burndy PSE36C-2 36-pin PCB edge connector. The pin assignment is shown below.

Table 2-6. Memory Port Connector Pin Assignments

Pin	Symbol	Signal Name
1	GND	Ground
2	$\overline{RD}$	Read
3	$\overline{WR}$	Write
4	$\overline{CSV}$	System reset
5	A0	Address bit 0
6	A1	Address bit 1
7	A2	Address bit 2
8	A3	Address bit 3
9	A4	Address bit 4
10	A5	Address bit 5
11	A6	Address bit 6
12	A7	Address bit 7
13	A8	Address bit 8
14	A9	Address bit 9
15	A10	Address bit 10
16	A11	Address bit 11
17	A12	Address bit 12
18	A13	Address bit 13
19	A14	Address bit 14
20	A15	Address bit 15
21	$\overline{CSMC1}$ *	Memory card chip select 1
22	$\overline{CSMC2}$ †	Memory card chip select 2
23	AD0	Address/data bit 0
24	AD1	Address/data bit 1
25	AD2	Address/data bit 2
26	AD3	Address/data bit 3
27	AD4	Address/data bit 4
28	AD5	Address/data bit 5
29	AD6	Address/data bit 6
30	AD7	Address/data bit 7
31	$\overline{CSV}$	System reset
32	V_{ni}	NiCd battery voltage (output)
33	V_{mcs}	Lithium battery voltage (input)
34	V_{ad}	Recharger DC voltage (output)
35	V_{cc}	Supply voltage
36	GND	Ground

* $\overline{CSMC1}$ is active when A19, A18, and A16 = 0 and A17 = 1 ($\overline{CSMC1} = \overline{A19} \cdot \overline{A18} \cdot A17 \cdot \overline{A16}$).

† $\overline{CSMC2}$ is active when A19 and A18 = 0, and A17 and A16 = 1 ($\overline{CSMC2} = \overline{A19} \cdot \overline{A18} \cdot A17 \cdot A16$).

External Bus Connector Specifications

The external bus connector is located on the underside of the HP-94 behind a hard plastic port cover. The connector is a JAE PICL-40S-ST. Connection to the external bus connector can be made using a JAE PICL-40P-ST connector. The pin assignment is shown in the following table. Pin 1 is marked on the connector body. The odd-numbered pins are on the outer row (toward the outer edge of the HP-94 case), and the even-numbered pins are on the inner row.

Table 2-7. External Bus Connector Pin Assignments

Pin	Symbol	Signal Name
1	V_{ni}	NiCd battery voltage
2	V_{ni}	NiCd battery voltage
3	V_{cc}	Supply voltage
4	GND	Ground
5	NC	Not connected
6	NC	Not connected
7	NC	Not connected
8	DT/$\overline{R}$	Buffer read/write
9	DEBUG	Connected to ground
10	NC	No connection
11	IRQFK	Reserved interrupt request 2
12	IRQPR	Reserved interrupt request 1
13	IO/$\overline{M}$	IO/Memory
14	ALE	Address latch enable
15	CLK	CPU clock
16	AS16	Address/status bit 16
17	AS17	Address/status bit 17
18	AS18	Address/status bit 18
19	AS19	Address/status bit 19
20	RESET	System clocked reset
21	AD0	Address/data bit 0
22	AD1	Address/data bit 1
23	AD2	Address/data bit 2
24	AD3	Address/data bit 3
25	AD4	Address/data bit 4
26	AD5	Address/data bit 5
27	AD6	Address/data bit 6
28	AD7	Address/data bit 7
29	A15	Address bit 15
30	A14	Address bit 14
31	A13	Address bit 13
32	A12	Address bit 12
33	A11	Address bit 11
34	A10	Address bit 10
35	A9	Address bit 9
36	A8	Address bit 8
37	$\overline{WR}$	Write
38	$\overline{RD}$	Read
39	GND	Ground
40	GND	Ground

Earphone Connector Specifications

The earphone jack accepts a 3.5 mm miniature plug, with an overall length of less than 12 mm. A 0.125 inch diameter miniature plug will also fit, but will tend to have a lower insertion and removal force. Most standard earphones will connect properly to the HP-94 both mechanically and electrically. Some variation in audio output volume will occur between various earphone manufacturers.

Battery Pack Connector Specifications

The battery pack uses two AMP 42827-1 brass contacts. The HP-94 mates these contacts with custom nickel-plated brass pins that are 2.31 mm (0.091 in) nominal diameter, 6.3 mm (0.25 in) long, and 6.0 mm (0.24 in) center spacing.

3

Environmental Specifications

3

Environmental Specifications

Below are the environmental specifications for the HP-94.

Table 3-1. Environmental Specifications

Parameter	Min	Max	Units	Comments
Operating Temperature	0	55	°C	+32 to 131 °F
Storage Temperature	-40	65	°C	-40 to 149 °F
Operating Humidity	0	95	%RH	At 40 °C
Vibration	3.4 g rms, 5 to 500 Hz random vibration, 10 minutes per axis			
	Swept sine, 1 g, 5 to 500 Hz, 10 minutes dwell at resonance			
Shock	3 ms, 1/2 sine wave, 228 g, 6 axes			

4

Accessory Specifications

Contents

Chapter 4 Accessory Specifications

4

Accessory Specifications

The principal HP-94 hardware accessories (at the time of printing) are listed below. This accessory list does not include any software documentation, development tools, or utilities. For a complete list of all HP-94 accessories and support items, please refer to the current HP-94 price list, available at all HP sales offices. This chapter will describe only accessories listed in the table below.

Table 4-1. HP-94 Hardware Accessories

Model No.	Description
HP 82411A	40K RAM Card
HP 82412A	32K-128K ROM/EPROM Card
HP 82430A	Rechargeable NiCd Battery Pack
HP 82431A	U.S./Canada Recharger
HP 82431AB *	Europe Recharger
HP 82431AG *	Australia Recharger
HP 82431AU *	U.K. Recharger
HP 82470A	RS-232-C Level Converter
HP 82433A	HP-94 to Modem Cable
HP 82434A	HP-94 to Printer Cable
HP 82435A	HP-94 to Level Converter Cable
HP 82436A	HP-94 to Vectra Cable
HP 24542G	Vectra or IBM PC/AT to Level Converter Cable
HP 17255D	IBM PC or PC/XT to Level Converter Cable
HP 39961D	Smart Wand - Low Resolution
HP 39963D	Smart Wand - General Purpose
HP 39965D	Smart Wand - High Resolution

* The foreign versions of the recharger (Europe, Australia, and U.K.) are not available at the time this document was printed.

40K RAM Card Specifications

Pin assignments for the HP 82411A 40K RAM Card are described in the "Mechanical Specifications" chapter. The RAM card uses the same Toshiba RAM (TC5565FL-15L) as is used in the HP-94. A CR-2032 (or equivalent) lithium battery is required to provide battery backup for the RAM card.

ROM/EPROM Card Specifications

Pin assignments for the HP 82412A ROM/EPROM Card are described in the "Mechanical Specifications" chapter.

The ROM/EPROM Card has sockets for up to three 32 Kbyte (256 Kbit) ROMs or EPROMs, or up to two 64 Kbyte (512 Kbit) ROMs or EPROMs or their equivalents. There is a socketed jumper on the card that allows selection of the different sizes. The generic names for these ICs are 27C256 for the 32 Kbyte and 27C512 for the 64 Kbyte ICs. The CMOS version of the ROMs or EPROMs must be used. The NMOS versions require more current than is guaranteed by the HP-94. The EPROMs cannot be programmed while in the ROM/EPROM card, but must be programmed in an external EPROM programmer.

A list of the required specifications is provided below to assist in selecting the appropriate parts.

Table 4-2. ROM and EPROM Specifications

Parameter	Min	Max	Units	Comments
Operating Voltage	4.5	5.5	Vdc	6.0 V is recommended *
Operating Temperature	-10	+65	°C	+14 to 149 °F
Access Time		250	ns	All parts ≤ 250 ns

* Several manufacturers (including Intel) offer EPROMs with extended operating voltage range.

The manufacturers that make correct size ROMs and EPROMs for use with the ROM/EPROM card and their part designations as of this printing are listed below. You should verify operating voltage, temperature, and speed with the manufacturer before making a final selection.

Table 4-3. ROM and EPROM Manufacturers

Manufacturer	32K IC		64K IC	
	EPROM	ROM	EPROM	ROM
Advanced Micro Devices	Am27C256	—	Am27C512	—
Fujitsu	MBM27C256	MB83256	MBM27C512	MB83512
Hitachi	HN27C256	HN613256P	—	—
Intel	27C256	—	27C512	—
Motorola	MCM67256	—	MCM67512	
National Semiconductor	NMC27C256	—	NMC27C512	—
NEC	μPD27C256	μPD23C256E	μPD27C512	μPD23C512
Texas Instruments	TMS27C256	TMS47C256	TMS27C512	TMS47C512
Toshiba	TC57256	TMM53257P	—	—

Battery Pack Specifications

The HP-94 uses the HP 82430A Rechargeable Battery Pack. When fully charged, the battery pack has approximately 900 milliamp-hours (mAh) of usable charge. The battery pack is charged whenever an HP 82431 recharger (HP 82431A/AB/AG/AU) is connected to the HP-94. Charging times and currents when charged using one of the HP 82431 rechargers are shown below.

Table 4-4. HP 82430A Rechargeable Battery Pack Specifications

Parameter	Symbol	Min	Typical	Max	Units	Comments
Capacity			900		mAh	
Charging Time	T_{ch}	6	10	14	hr	*
Charging Current	I_{ch}		150 †		mA	Pack attached to HP-94
Charging Current	I_{ch}			200 ‡	mA	Pack detached from HP-94

* The battery pack charging time is independent of HP-94 operating mode. Sufficient current is provided to operate the HP-94 and its principal accessories as well as fully charge the battery pack.

Charging times in excess of 18 hours are not recommended. Extended charging time may reduce the life of the battery pack. It is recommended that periodic "deep discharge - full recharge" cycles be performed to insure that maximum life and charge retention performance of the battery pack is maintained.

† The battery is connected to the recharger through a 2.7 Ω current-limiting resistor in series with a Schottky blocking diode. The actual charging current will vary as the battery pack voltage increases from the discharged state to the full charged state.

‡ Charging at currents greater than 200 mA for extended periods of time may damage the battery pack.

The battery pack contains four 2/3 C NiCd batteries completely enclosed in a detachable battery housing. All NiCd batteries are capable of extremely high short circuit currents. A thermal protector is built into the battery pack to prevent a constant short circuit condition. Since this circuit is temperature-sensitive, ambient conditions at or above its 75 °C temperature rating will cause a temporary open circuit in the battery pack. The HP-94 will then behave as if no battery pack is connected. When the short circuit or high temperature condition is removed, the battery pack short circuit protector will again close and the battery pack will continue normal operation.

WARNING **Never connect multiple battery packs in parallel while charging. Each individual pack should be blocked with a diode to prevent short circuit current from a failed cell from flowing into good cells of other packs.**

The battery pack connector specifications are described in the "Mechanical Specifications" chapter.

Guidelines for Using Rechargeable Batteries

The following is usage information and cautions about using rechargeable batteries.

CAUTION To avoid damage to the handheld computer, use only the batteries and recharger designated by Hewlett-Packard for the computer. Also, do not allow the batteries to discharge beyond their available capacity — recharge as soon as possible after the low battery indication appears. Allowing rechargeable batteries to discharge beyond their maximum limit can damage the batteries.

- Recharging batteries before they are low may eventually decrease their charging capacity.
- Do not overcharge the batteries by allowing them to recharge for longer than the recommended time. Shorter charging times will reduce the operating time before recharging is required, but will not harm the batteries.
- Do not leave the recharger permanently connected to the machine. Doing so decreases the useful life of the batteries.
- Do not use the recharger if it appears to have loose contacts, a cracked housing, or a damaged cord.
- Properly dispose of the batteries when they no longer adequately hold a charge or when they appear damaged.

WARNING **To prevent injury, keep all batteries out of the reach of children and properly dispose of exhausted batteries. Do not mutilate or puncture batteries, and do not dispose of them in fire. Exposure to excessive heat can cause release of toxic fumes or explosion.**

Recharger Specifications

The HP 82431 Recharger (HP 82431A/AB/AG/AU) supplies charging current to the HP-94's NiCd battery pack. The recharger is designed to supply sufficient current to charge the batteries even while the HP-94 is operating.

Table 4-5. HP 82431 Recharger Specifications

Parameter	Symbol	Min	Typical	Max	Units	Comments
Input Voltage	V_{ac}	108	120	132	Vac	HP 82431A
		198	220	242	Vac	HP 82431AB
		216	240	264	Vac	HP 82431AG/AU
Input Current	I_{ac}			80	mA	HP 82431A
				40	mA	HP 82431AB/AG/AU
Input Frequency	I_{fr}	57.5	60	62.5	Hz	HP 82431A
		47.5	50	55	Hz	HP 82431AB/AG/AU
Output Voltage	V_{rch}	6.2		6.7	Vdc	
Output Current	I_{rch}			400	mA	*

* Refer to "Battery Pack Specifications" for details about the charging current actually supplied to the battery pack.

Level Converter Specifications

The HP-94 serial port outputs CMOS logic levels (refer to "Electrical Specifications"). Some RS-232 devices require that proper RS-232 voltage levels be provided for their serial interfaces to operate properly. These devices require the use of the HP 82470A RS-232-C Level Converter.

The level converter modifies the 0 to V_{cc} voltage level outputs from the HP-94 serial port to ±9 V EIA RS-232-C voltage levels. Additionally, the level converter's 25-pin connector inputs and outputs meet all RS-232 timing and load specifications. RS-232 voltage level inputs to the level converter's 25-pin connector are internally shifted to the 0 to V_{cc} range the HP-94 expects, and then are output to the HP-94 using the 15-pin connector.

When the serial port is disabled, the control lines are turned off (set to 0 volts). This is different than most AC-powered serial devices, in which the control lines are high (-3 volts or less) because the serial port is powered whenever the device is on.

Connection is made between the HP-94 and the level converter using an HP 82435A 1/4 meter cable. The level converter is powered by the HP-94 using pin 9 of the serial port (V_{rs}). The output voltage V_{rs} is activated under program control when the serial port is enabled (refer to the "Serial Port" chapter in the operating system section of this manual). Typical power consumption by the level converter is 25 mA when active with a standard RS-232-C load at 9600 baud and V_{cc} = 6.0 volts ($V_{rs}=V_{cc}$-0.1 V).

Below are the pin assignments for both the 15- and 25-pin connectors on the level converter.

Table 4-6. HP 82470A RS-232-C Level Converter Pin Assignments

25-Pin Female Connector			15-Pin Female Connector		
Signal Name	**Symbol**	**Pin No.**	**Pin No.**	**Symbol**	**HP-94 Signal Name**
Frame Ground	FG	Housing	Housing	FG	Frame Ground
Transmitted Data	TxD	2	2	TxD	Transmitted Data
Received Data	RxD	3	3	RxD	Received Data
Request To Send	RTS	4	4	RTS	Request To Send
Clear To Send	CTS	5	5	CTS	Clear To Send
Data Set Ready	DSR	6	6	DSR	Data Set Ready
Signal Ground	SG	7	7	SG	Signal Ground
Data Carrier Detect	DCD	8	8	DCD	Data Carrier Detect
Not Connected	NC	9	9	V_{rs}	Switched V_{cc} *
Data Terminal Ready	DTR	20	15	DTR	Data Terminal Ready

* HP 82470A level converter power.

When to Use the Level Converter

RS-232-C specifications require that input signal levels at the input of a device be greater than +3 or less than -3 volts. RS-232 output voltages experience a greater voltage swing to prevent signal degradation and line noise from interfering with communication signals. However, many available line receivers do not actually require voltage swings of these levels. The HP-94 system can take advantage of this by not requiring that the level converter be used when communicating with these devices.

The HP-94 will switch its RS-232 outputs between CMOS logic levels, where V_{cc} will be between 4.5 and 6.0 volts. This provides logic low levels of less than 0.2 volts and logic high levels of greater than V_{cc}-0.2 volts. Thus, any line receiver that will respond with high-to-low and low-to-high transitions in this range of logic 0/1 values will not need to have true RS-232 levels at its inputs to properly detect the logic level.

The line receivers that can communicate directly with the HP-94 (that is, no level converter required) are listed below. Certain parts listed must be operated in the specified mode or configuration, so special attention must be paid to the comments.

Table 4-7. Line Receivers That Do Not Require Level Converter

Part Number	Manufacturer	Comments
1489	National Semiconductor Motorola *	Response (threshold) control must be open
75189	Texas Instruments	Response (threshold) control must be open
75154	Texas Instruments	Response (threshold) control must be open
MAX232	Maxim	
MC145406	Motorola	
74HC14	many manufacturers	

* 1489-compatible parts from other manufacturers will also work.

CAUTION When using the HP-94 system without a level converter, special care must be taken to ensure that the interconnection cables are sufficiently short to prevent signal degradation. It is recommended that all communications cable for use with the HP-94 that do not use the level converter be less than 3 meters in length.

Cables

There are several cables available to allow configuration of the HP-94 in a system. The connections for each of these cables are provided in the tables that follow in this section. Cable lengths are 1 meter unless specified otherwise.

Modem Cable

The HP 82433A cable is used to connect the HP-94 to modems that do not require a level converter. It is specifically designed for use with Hayes Smartmodems, but is usable with many other modems as well.

Table 4-8. HP-94 to Modem Cable

HP-94 15-Pin Male Connector			Modem 25-Pin Male Connector		
Signal Name	**Symbol**	**Pin No.**	**Pin No.**	**Symbol**	**Direction**
Frame Ground	FG	Housing	Housing	AA	N/A
Transmitted Data	TxD	2	2	BA	To Modem
Received Data	RxD	3	3	BB	From Modem
Request To Send	RTS	4	4	CA *	To Modem
Clear To Send	CTS	5	5	CB	From Modem
Data Set Ready	DSR	6	6	CC	From Modem
Signal Ground	SG	7	7	AB	N/A
Data Carrier Detect	DCD	8	8	CF	From Modem
Data Terminal Ready	DTR	15	20	CD	To Modem

* Hayes Smartmodems do not implement this line.

The HP-94 has receive all the necessary approvals for connecting to modems in the U.S. Some countries require that the product and its interface cable be approved prior to connecting to a modem. Contact your local Hewlett-Packard sales office to verify that the HP-94 is approved for your specific location.

Printer Cable

The HP 82434A cable is used to connect the HP-94 to RS-232-C printers that do not require a level converter. It is specifically designed for use with Hewlett-Packard ThinkJet printers (HP 2225D), but is usable with many other printers as well.

Table 4-9. HP-94 to Printer Cable

HP-94 15-Pin Male Connector			Printer 25-Pin Male Connector		
Signal Name	**Symbol**	**Pin No.**	**Pin No.**	**Symbol**	**Signal Name**
Frame Ground	FG	Housing	Housing	AA	Protective Ground
Transmitted Data	TxD	2	3	BB	Received Data
Received Data	RxD	3	2	BA	Transmitted Data
Request To Send	RTS	4	5	CB *	Clear To Send
Clear To Send	CTS	5	4	CA	Request To Send
Data Set Ready	DSR	6	20	CD	Data Terminal Ready
Signal Ground	SG	7	7	AB	Signal Ground

* Many printers (including the Hewlett-Packard ThinkJet) do not implement this line.

Level Converter Cable

When using the level converter, an HP 82435A 1/4-meter cable is required. This cable provides a straight-through connection between the HP-94 and the level converter.

Table 4-10. HP-94 to Level Converter Cable

HP-94 15-Pin Male Connector			Level Converter 15-Pin Female Connector		
Signal Name	**Symbol**	**Pin No.**	**Pin No.**	**Symbol**	**Signal Name**
Frame Ground	FG	Housing	Housing	FG	Frame Ground
Transmitted Data	TxD	2	2	TxD	Transmitted Data
Received Data	RxD	3	3	RxD	Received Data
Request To Send	RTS	4	4	RTS	Request To Send
Clear To Send	CTS	5	5	CTS	Clear To Send
Data Set Ready	DSR	6	6	DSR	Data Set Ready
Signal Ground	SG	7	7	SG	Signal Ground
Data Carrier Detect	DCD	8	8	DCD	Data Carrier Detect
Switched V_{cc}	V_{rs} *	9	9	V_{rs} *	Switched V_{cc}
Data Terminal Ready	DTR	15	15	DTR	Data Terminal Ready

* HP 82470A level converter power.

Vectra Cable

The HP 82436A 2-meter cable is used whenever direct communication between the HP-94 and a 9-pin serial port on an HP Vectra personal computer. Each of the two Vectra serial interfaces has one 9-pin port: the HP 24540A Serial/Parallel Interface, and the HP 24541A Dual Serial Interface. HP supplies no cables that connect the HP-94 directly to the 25-pin port on the Vectra Dual Serial Interface.

Table 4-11. HP-94 to Vectra Cable

Vectra 9-Pin Female Connector			HP-94 15-Pin Male Connector		
Signal Name	**Symbol**	**Pin No.**	**Pin No.**	**Symbol**	**Signal Name**
Protective Ground	AA	Housing	Housing	FG	Frame Ground
Received Data	BB	2	2	TxD	Transmitted Data
Transmitted Data	BA	3	3	RxD	Received Data
Data Terminal Ready	CD	4	5 *	CTS	Clear to Send
Data Terminal Ready	CD	4	6 *	DSR	Data Set Ready
Signal Ground	AB	5	7	SG	Signal Ground
Data Set Ready	CC	6 *	15	DTR	Data Terminal Ready
Clear to Send	CB	8 *	15	DTR	Data Terminal Ready

* Pins 6 and 8 are tied together on the 9-pin connector, and pins 5 and 6 are tied together on the 15-pin connector.

Vectra or IBM PC/AT to Level Converter Cable

The HP-94 can communicate directly with the HP Vectra computer through the HP 82436A cable, without using a level converter. For applications that require extended cable lengths or desire the level converter option, the HP 24542G Serial Printer/Plotter Cable can be used. When communicating with the IBM PC/AT, a level converter is required, and this cable must be used. The level converter is then connected to the HP-94 using the HP 82435A cable.

Table 4-12. Vectra or IBM PC/AT to Level Converter Cable

Vectra or IBM PC/AT 9-Pin Female Connector			Level Converter 25-Pin Male Connector		
Signal Name	**Symbol**	**Pin No.**	**Pin No.**	**Symbol**	**Signal Name**
Protective Ground	AA	Housing	Housing	FG	Frame Ground
Data Carrier Detect	CF	1	4	RTS	Request To Send
Received Data	BB	2	2	TxD	Transmitted Data
Transmitted Data	BA	3	3	RxD	Received Data
Data Terminal Ready	CD	4	5 *	CTS	Clear To Send
Data Terminal Ready	CD	4	6 *	DSR	Data Set Ready
Signal Ground	AB	5	7	SG	Signal Ground
Data Set Ready	CC	6 *	20	DTR	Data Terminal Ready
Request To Send	CA	7	8	DCD	Data Carrier Detect
Clear To Send	CB	8 *	20	DTR	Data Terminal Ready

* Pins 6 and 8 are tied together on the 9-pin connector, and pins 5 and 6 are tied together on the 25-pin connector.

IBM PC or PC/XT to Level Converter Cable

When using an IBM PC or PC/XT to communicate with the HP-94, a level converter is required. The HP 17255D cable connects the 25-pin IBM serial port connector to the 25-pin connector on the level converter. The level converter is then connected to the HP-94 using the HP 82435A cable.

Table 4-13. IBM PC or PC/XT to Level Converter Cable

IBM PC or PC/XT 25-Pin Female Connector			Level Converter 25-Pin Male Connector		
Signal Name	**Symbol**	**Pin No.**	**Pin No.**	**Symbol**	**Signal Name**
Frame Ground	FG	Housing *	Housing *	FG	Frame Ground
Transmitted Data	TxD	2	3	RxD	Received Data
Received Data	RxD	3	2	TxD	Transmitted Data
Clear To Send	CTS	5 †	20	DTR	Data Terminal Ready
Data Set Ready	DSR	6 †	20	DTR	Data Terminal Ready
Signal Ground	SG	7	7	SG	Signal Ground
Data Terminal Ready	DTR	20	5 †	CTS	Clear To Send
Data Terminal Ready	DTR	20	6 †	DSR	Data Set Ready

* Pin 1 is connected to frame ground (housing) on both connectors.
† Pins 5 and 6 are tied together on both connectors.

Bar Code Readers

The primary bar code readers for the HP-94 are the three HP Smart Wands: HP 39961D (low resolution), HP 39963D (general purpose), and HP 39965D (high resolution). Contact your sales office for complete literature and specifications for these wands.

Connecting the Serial Port to a Smart Wand

HP Smart Wands can be configured in one of two ways:

- By scanning bar code configuration menus (optical configuration)
- By sending configuration escape sequences to the Smart Wand

When a Smart Wand is connected to the bar code port, only the first approach is available because the bar code port is read-only. The second approach is available if the Smart Wand can be connected to the serial port. To support this use, Hewlett-Packard supplies a low-level bar code handler with the *HP-94 Software Development System* called HNSP that allows "smart" bar code scanning devices to be connected to the serial port.

HP Smart Wands are supplied with a 5-pin, 240° circular DIN connector, but at this printing are not available with a 15-pin D-type connector that would connect to the serial port. Below are the connections for a cable that will connect the serial port to a Smart Wand. This cable is not available from Hewlett-Packard — the connections are provided to allow a developer to make the cable.

Table 4-14. HP-94 Serial Port to Smart Wand Cable

HP-94 15-Pin Male Connector			Smart Wand 5-Pin or 6-Pin Female Connector		
Signal Name	**Symbol**	**Pin No.**	**Pin No.**	**Symbol**	**Signal Name**
Frame Ground	FG	Housing *	Housing *	FG	Frame Ground
Transmitted Data	TxD	2	4	RxD	Received Data
Received Data	RxD	3	2	TxD	Transmitted Data
Request To Send	RTS	4 †	—	NC	Not Connected
Clear To Send	CTS	5 †	—	NC	Not Connected
Signal Ground	SG	7	3	SG	Signal Ground
Switched V_{cc}	V_{rs} ‡	9	1	V_{rs} ‡	Switched V_{cc}

* The shield or braid must be connected to frame ground (housing).

† Pins 4 and 5 are tied together on the 15-pin connector.

‡ HP Smart Wand power.

5

Data Sheets

5

Data Sheets

This chapter contains copies of manufacturer's data sheets for the following four ICs:

- NEC μPD70108 (V20) Microprocessor
- OKI MSM82C51A Universal Asynchronous Receiver Transmitter (UART)
- Hitachi HD61102A LCD Column Driver
- Epson RTC-58321 Real-Time Clock

These data sheets provide reference information for developers whose application interacts directly with the IC, independent of the HP-94 operating system. Refer to the appropriate chapters in the "Operating System" for information about how these ICs are used in the HP-94, what I/O control registers are associated with each IC, and what built-in software is available already to control them.

NEC μPD70108 (V20) Microprocessor Data Sheet

NEC

NEC Electronics Inc.

μPD70108 (V20) HIGH-PERFORMANCE 16-BIT MICROPROCESSOR

Revision 3

November 1985

Description

The μPD70108 (V20) is a CMOS 16-bit microprocessor with internal 16-bit architecture and an 8-bit external data bus. The μPD70108 instruction set is a superset of the μPD8086/8088; however, mnemonics and execution times are different. The μPD70108 additionally has a powerful instruction set including bit processing, packed BCD operations, and high-speed multiplication/division operations. The μPD70108 can also execute the entire 8080 instruction set and comes with a standby mode that significantly reduces power consumption. It is software-compatible with the μPD70116 16-bit microprocessor.

Features

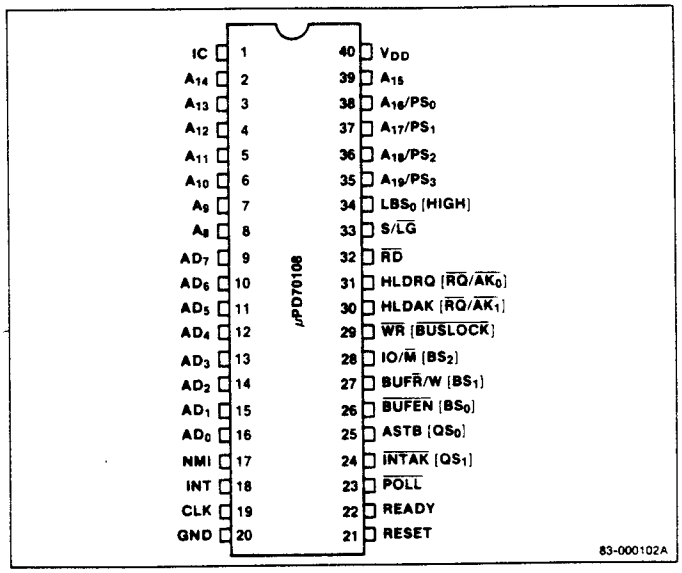
- ☐ Minimum instruction execution time: 250 ns (at 8 MHz)
- ☐ Maximum addressable memory: 1 Mbyte
- ☐ Abundant memory addressing modes
- ☐ 14 x 16-bit register set
- ☐ 101 instructions
- ☐ Instruction set is a superset of μPD8086/8088 instruction set
- ☐ Bit, byte, word, and block operations
- ☐ Bit field operation instructions
- ☐ Packed BCD instructions
- ☐ Multiplication/division instruction execution time: 4 μs to 6 μs (at 8 MHz)
- ☐ High-speed block transfer instructions: 1 Mbyte/s (at 8 MHz)
- ☐ High-speed calculation of effective addresses: 2 clock cycles in any addressing mode
- ☐ Maskable (INT) and nonmaskable (NMI) interrupt inputs
- ☐ IEEE-796 bus compatible interface
- ☐ 8080 emulation mode
- ☐ CMOS technology
- ☐ Low-power consumption
- ☐ Low-power standby mode
- ☐ Single power supply
- ☐ 5 MHz, 8 MHz or 10 MHz clock

Ordering Information

Part Number	Package Type	Max Frequency of Operation
μPD70108C-5	40-pin plastic DIP	5 MHz
μPD70108C-8	40-pin plastic DIP	8 MHz
μPD70108D-5	40-pin ceramic DIP	5 MHz
μPD70108D-8	40-pin ceramic DIP	8 MHz
μPD70108D-10	40-pin ceramic DIP	10 MHz
μPD70108G-5	52-pin flat pack	5 MHz
μPD70108G-8	52-pin flat pack	8 MHz
μPD70108L-5	44-pin PLCC	5 MHz
μPD70108L-8	44-pin PLCC	8 MHz

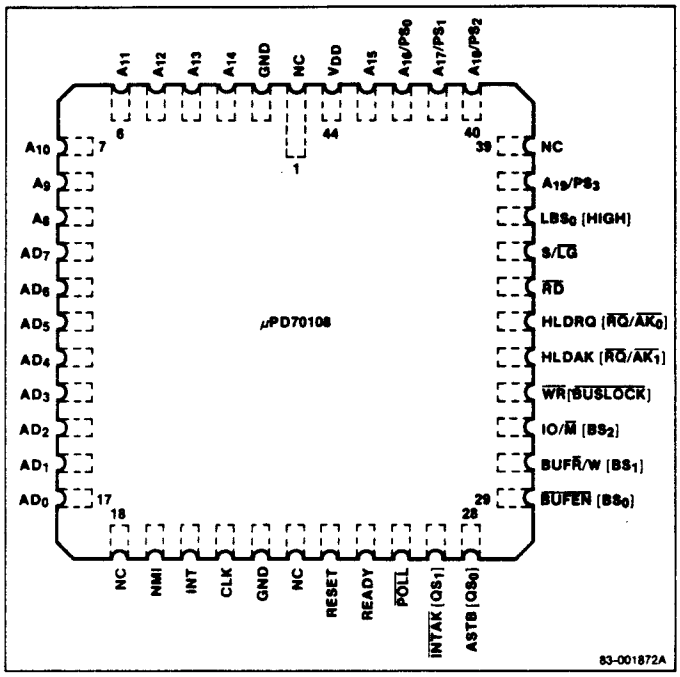
Pin Configurations

40-Pin Plastic DIP/Cerdip

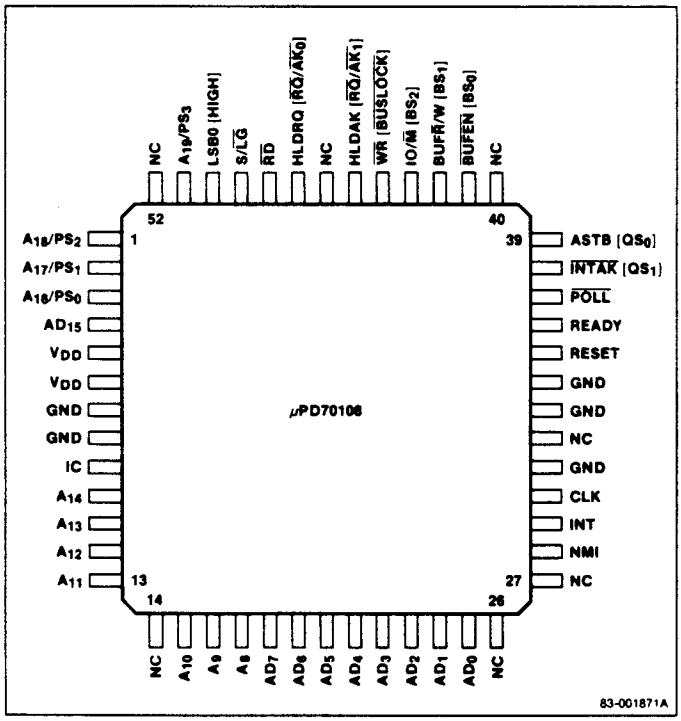


Pin Configurations (cont)

44-Pin Plastic Leadless Chip Carrier (PLCC)



52-Pin Plastic Miniflat



Pin Identification

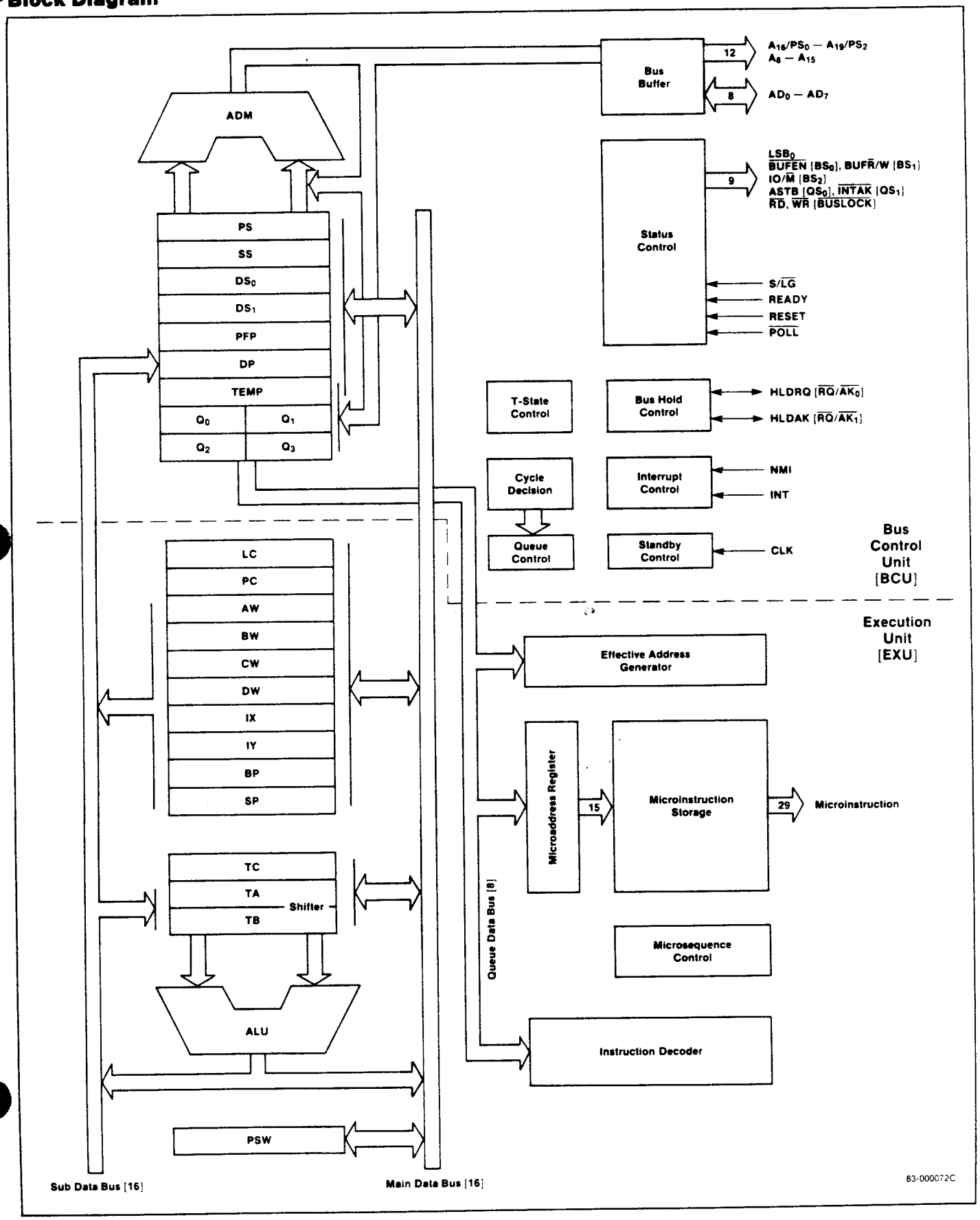
No.	Symbol	Direction	Function
1	IC*		Internally connected
2 - 8	A_{14} - A_8	Out	Address bus, middle bits
9 - 16	AD_7 - AD_0	In/Out	Address/data bus
17	NMI	In	Nonmaskable interrupt input
18	INT	In	Maskable interrupt input
19	CLK	In	Clock input
20	GND		Ground potential
21	RESET	In	Reset input
22	READY	In	Ready input
23	$\overline{POLL}$	In	Poll input
24	$\overline{INTAK}$ (QS_1)	Out	Interrupt acknowledge output (queue status bit 1 output)
25	ASTB (QS_0)	Out	Address strobe output (queue status bit 0 output)
26	$\overline{BUFEN}$ (BS_0)	Out	Buffer enable output (bus status bit 0 output)
27	$BUF\overline{R}/W$ (BS_1)	Out	Buffer read/write output (bus status bit 1 output)
28	$IO/\overline{M}$ (BS_2)	Out	Access is I/O or memory (bus status bit 2 output)
29	$\overline{WR}$ ($\overline{BUSLOCK}$)	Out	Write strobe output (bus lock output)
30	HLDAK ($\overline{RQ}/\overline{AK}_1$)	Out (In/Out)	Hold acknowledge output, (bus hold request input/ acknowledge output 1)
31	HLDRQ ($\overline{RQ}/\overline{AK}_0$)	In (In/Out)	Hold request input (bus hold request input/ acknowledge output 0)
32	$\overline{RD}$	Out	Read strobe output
33	$S/\overline{LG}$	In	Small-scale/large-scale system input
34	LBS_0 (HIGH)	Out	Latched bus status output 0 (always high in large-scale systems)
35 - 38	A_{19}/PS_3 - A_{16}/PS_0	Out	Address bus, high bits or processor status output
39	A_{15}	Out	Address bus, bit 15
40	V_{DD}		Power supply

Notes: * IC should be connected to ground.

Where pins have different functions in small- and large-scale systems, the large-scale system pin symbol and function are in parentheses.

Unused input pins should be tied to ground or V_{DD} to minimize power dissipation and prevent the flow of potentially harmful currents.

Block Diagram



Pin Functions

Some pins of the μPD70108 have different functions according to whether the microprocessor is used in a small- or large-scale system. Other pins function the same way in either type of system.

A_{15} - A_8 [Address Bus]
For small- and large-scale systems.

The CPU uses these pins to output the middle 8 bits of the 20-bit address data. They are three-state outputs and become high impedance during hold acknowledge.

AD_7 - AD_0 [Address/Data Bus]
For small- and large-scale systems.

The CPU uses these pins as the time-multiplexed address and data bus. When high, an AD bit is a one; when low, an AD bit is a zero. This bus contains the lower 8 bits of the 20-bit address during T1 of the bus cycle and is used as an 8-bit data bus during T2, T3, and T4 of the bus cycle.

Sixteen-bit data I/O is performed in two steps. The low byte is sent first, followed by the high byte. The address/data bus is a three-state bus and can be at a high or low level during standby mode. The bus will be high impedance during hold and interrupt acknowledge.

NMI [Nonmaskable Interrupt]
For small- and large-scale systems.

This pin is used to input nonmaskable interrupt requests. NMI cannot be masked by software. This input is positive edge triggered and must be held high for five clocks to guarantee recognition. Actual interrupt processing begins, however, after completion of the instruction in progress.

The contents of interrupt vector 2 determine the starting address for the interrupt-servicing routine. Note that a hold request will be accepted even during NMI acknowledge.

This interrupt will cause the μPD70108 to exit the standby mode.

INT [Maskable Interrupt]
For small- and large-scale systems.

This pin is an interrupt request that can be masked by software.

INT is active high level and is sensed during the last clock of the instruction. The interrupt will be accepted if the interrupt enable flag IE is set. The CPU outputs the INTAK signal to inform external devices that the interrupt request has been granted. INT must be asserted until the interrupt acknowledge is returned.

If NMI and INT interrupts occur at the same time, NMI has higher priority than INT and INT cannot be accepted. A hold request will be accepted during INT acknowledge.

This interrupt causes the μPD70108 to exit the standby mode.

CLK [Clock]
For small- and large-scale systems.

This pin is used for external clock input.

RESET [Reset]
For small- and large-scale systems.

This pin is used for the CPU reset signal. It is an active high level. Input of this signal has priority over all other operations. After the reset signal input returns to a low level, the CPU begins execution of the program starting at address FFFF0H.

In addition to causing normal CPU start, RESET input will cause the μPD70108 to exit the standby mode.

READY [Ready]
For small- and large-scale systems.

When the memory or I/O device being accessed cannot complete data read or write within the CPU basic access time, it can generate a CPU wait state (Tw) by setting this signal to inactive (low level) and requesting a read/write cycle delay.

If the READY signal is active (high level) during either the T3 or Tw state, the CPU will not generate a wait state.

$\overline{\text{POLL}}$ [Poll]
For small- and large-scale systems.

The CPU checks this input upon execution of the $\overline{\text{POLL}}$ instruction. If the input is low, then execution continues. If the input is high, the CPU will check the $\overline{\text{POLL}}$ input every five clock cycles until the input becomes low again.

The $\overline{\text{POLL}}$ and READY functions are used to synchronize CPU program execution with the operation of external devices.

$\overline{\text{RD}}$ [Read Strobe]
For small- and large-scale systems.

The CPU outputs this strobe signal during data read from an I/O device or memory. The IO/$\overline{\text{M}}$ signal is used to select between I/O and memory.

The three-state output is held high during standby mode and enters the high-impedance state during hold acknowledge.

S/$\overline{\text{LG}}$ [Small/Large]
For small- and large-scale systems.

This signal determines the operation mode of the CPU. This signal is fixed at either a high or low level. When

this signal is a high level, the CPU will operate in small-scale system mode, and when low, in the large-scale system mode. A small-scale system will have at most one bus master such as a DMA controller device on the bus. A large-scale system can have more than one bus master accessing the bus as well as the CPU.

Pins 24 to 31 and pin 34 function differently depending on the operating mode of the CPU. Separate nomenclature is adopted for these signals in the two operating modes.

Pin No.	Function	
	S/$\overline{LG}$-high	S/$\overline{LG}$-low
24	$\overline{INTAK}$	QS_1
25	ASTB	QS_0
26	$\overline{BUFEN}$	BS_0
27	BUF$\overline{R}$/W	BS_1
28	IO/$\overline{M}$	BS_2
29	$\overline{WR}$	$\overline{BUSLOCK}$
30	HLDAK	$\overline{RQ}/\overline{AK}_1$
31	HLDRQ	$\overline{RQ}/\overline{AK}_0$
34	LBS_0	Always high

$\overline{INTAK}$ [Interrupt Acknowledge]
For small-scale systems.

The CPU generates the $\overline{INTAK}$ signal low when it accepts an INT signal.

The interrupting device synchronizes with this signal and outputs the interrupt vector to the CPU via the data bus (AD_7 - AD_0).

ASTB [Address Strobe]
For small-scale systems.

The CPU outputs this strobe signal to latch address information at an external latch.

ASTB is held at a low level during standby mode and hold acknowledge.

$\overline{BUFEN}$ [Buffer Enable]
For small-scale systems.

This is used as the output enable signal for an external bidirectional buffer. The CPU generates this signal during data transfer operations with external memory or I/O devices or during input of an interrupt vector.

This three-state output is held high during standby mode and enters the high-impedance state during hold acknowledge.

BUF$\overline{R}$/W [Buffer Read/Write]
For small-scale systems.

The output of this signal determines the direction of data transfer with an external bidirectional buffer. A high output causes transmission from the CPU to the external device; a low signal causes data transfer from the external device to the CPU.

BUF$\overline{R}$/W is a three-state output and becomes high impedance during hold acknowledge.

IO/$\overline{M}$ [IO/Memory]
For small-scale systems.

The CPU generates this signal to specify either I/O access or memory access. A high-level output specifies I/O and a low-level signal specifies memory.

IO/M's output is three state and becomes high impedance during hold acknowledge.

$\overline{WR}$ [Write Strobe]
For small-scale systems.

The CPU generates this strobe signal during data write to an I/O device or memory. Selection of either I/O or memory is performed by the IO/M signal.

This three-state output is held high during standby mode and enters the high-impedance state during hold acknowledge.

HLDAK [Hold Acknowledge]
For small-scale systems.

The HLDAK signal is used to indicate that the CPU accepts the hold request signal (HLDRQ). When this signal is a high level, the address bus, address/data bus, and the control lines become high impedance.

HLDRQ [Hold Request]
For small-scale systems.

This input signal is used by external devices to request the CPU to release the address bus, address/data bus, and the control bus.

LBS_0 [Latched Bus Status 0]
For small-scale systems.

The CPU uses this signal along with the IO/$\overline{M}$ and BUF$\overline{R}$/W signals to inform an external device what the current bus cycle is.

IO/$\overline{M}$	BUFR/$\overline{W}$	LBS_0	Bus Cycle
0	0	0	Program fetch
0	0	1	Memory read
0	1	0	Memory write
0	1	1	Passive state
1	0	0	Interrupt acknowledge
1	0	1	I/O read
1	1	0	I/O write
1	1	1	Halt

A_{19}/PS_3 - A_{16}/PS_0 [Address Bus/Processor Status]
For small- and large-scale systems.

These pins are time multiplexed to operate as an address bus and as processor status signals.

When used as the address bus, these pins are the high 4 bits of the 20-bit memory address. During I/O access, all 4 bits output data 0.

The processor status signals are provided for both memory and I/O use. PS_3 is always 0 in the native mode and 1 in 8080 emulation mode. The interrupt enable flag (IE) is pin on pin PS_2. Pins PS_1 and PS_0 indicate which memory segment is being accessed.

A_{17}/PS_1	A_{16}/PS_0	Segment
0	0	Data segment 1
0	1	Stack segment
1	0	Program segment
1	1	Data segment 0

The output of these pins is three state and becomes high impedance during hold acknowledge.

QS_1, QS_0 [Queue Status]
For large-scale systems.

The CPU uses these signals to allow external devices, such as the floating-point arithmetic processor chip, (μPD72091) to monitor the status of the internal CPU instruction queue.

QS_1	QS_0	Instruction Queue Status
0	0	NOP (queue does not change)
0	1	First byte of instruction
1	0	Flush queue
1	1	Subsequent bytes of instruction

The instruction queue status indicated by these signals is the status when the execution unit (EXU) accesses the instruction queue. The data output from these pins is therefore valid only for one clock cycle immediately following queue access. These status signals are provided so that the floating-point processor chip can monitor the CPU's program execution status and synchronize its operation with the CPU when control is passed to it by the FPO (Floating Point Operation) instructions.

BS_2 - BS_0 [Bus Status]
For large-scale systems.

The CPU uses these status signals to allow an external bus controller to monitor what the current bus cycle is.

The external bus controller decodes these signals and generates the control signals required to perform access of the memory or I/O device.

BS_2	BS_1	BS_0	Bus Cycle
0	0	0	Interrupt acknowledge
0	0	1	I/O read
0	1	0	I/O write
0	1	1	Halt
1	0	0	Program fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Passive state

The output of these signals is three state and becomes high impedance during hold acknowledge.

$\overline{BUSLOCK}$ [Bus Lock]
For large-scale systems.

The CPU uses this signal to secure the bus while executing the instruction immediately following the BUSLOCK prefix instruction, or during an interrupt acknowledge cycle. It is a status signal to the other bus masters in a multiprocessor system, inhibiting them from using the system bus during this time.

The output of this signal is three state and becomes high impedance during hold acknowledge. $\overline{BUSLOCK}$ is high during standby mode except if the HALT instruction has a BUSLOCK prefix.

$\overline{RQ}/\overline{AK}_1$, $\overline{RQ}/\overline{AK}_0$ [Hold Request/Acknowledge]
For large-scale systems.

These pins function as bus hold request inputs ($\overline{RQ}$) and as bus hold acknowledge outputs ($\overline{AK}$). $\overline{RQ}/AK_0$ has a higher priority than $\overline{RQ}/\overline{AK}_1$.

These pins have three-state outputs with on-chip pull-up resistors which keep the pin at a high level when the output is high impedance.

V_{DD} [Power Supply]
For small- and large-scale systems.

This pin is used for the +5 V power supply.

GND [Ground]
For small- and large-scale systems.

This pin is used for ground.

IC [Internally Connected]
This pin is used for tests performed at the factory by NEC. The μPD70108 is used with this pin at ground potential.

Absolute Maximum Ratings

$T_A = +25°C$

Power supply voltage, V_{DD}	−0.5 V to +7.0 V
Power dissipation, PD_{MAX}	0.5 W
Input voltage, V_I	−0.5 V to V_{DD} + 0.3 V
CLK input voltage, V_K	−0.5 V to V_{DD} + 1.0 V
Output voltage, V_O	−0.5 V to V_{DD} + 0.3 V
Operating temperature, T_{OPT}	−40°C to +85°C
Storage temperature, T_{STG}	−65°C to +150°C

Comment: Exposing the device to stresses above those listed in Absolute Maximum Ratings could cause permanent damage. The device is not meant to be operated under conditions outside the limits described in the operational sections of this specification. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Capacitance

$T_A = +25°C$, $V_{DD} = 0$ V

Parameter	Symbol	Limits Min	Limits Max	Unit	Test Conditions
Input capacitance	C_I		15	pF	fc = 1 MHz Unmeasured pins returned to 0 V
I/O capacitance	C_{IO}		15	pF	

DC Characteristics

μPD70108-5, $T_A = -40°C$ to +85°C, $V_{DD} = +5$ V ± 10%
μPD70108-8, μPD70108-10, $T_A = -10°C$ to +70°C, $V_{DD} = +5$ V ± 5%

Parameter	Symbol	Limits Min	Limits Typ	Limits Max	Unit	Test Conditions
Input voltage high	V_{IH}	2.2		V_{DD} + 0.3	V	
Input voltage low	V_{IL}	−0.5		0.8	V	
CLK input voltage high	V_{KH}	3.9		V_{DD} + 1.0	V	
CLK input voltage low	V_{KL}	−0.5		0.6	V	
Output voltage high	V_{OH}	0.7 x V_{DD}			V	$I_{OH} = -400$ μA
Output voltage low	V_{OL}			0.4	V	$I_{OL} = 2.5$ mA
Input leakage current high	I_{LIH}			10	μA	$V_I = V_{DD}$
Input leakage current low	I_{LIL}			−10	μA	$V_I = 0$ V
Output leakage current high	I_{LOH}			10	μA	$V_O = V_{DD}$
Output leakage current low	I_{LOL}			−10	μA	$V_O = 0$ V
Supply current	I_{DD}	70108-5	30	60	mA	Normal operation
		5 MHz	5	10	mA	Standby mode
		70108-8	45	80	mA	Normal operation
		8 MHz	6	12	mA	Standby mode
		70108-10	60	100	mA	Normal operation
		10 MHz	7	14	mA	Standby mode

AC Characteristics

μPD70108-5, $T_A = -40°C$ to $+85°C$, $V_{DD} = +5\ V \pm 10\%$
μPD70108-8, μPD70108-10, $T_A = -10°C$ to $+70°C$, $V_{DD} = +5\ V \pm 5\%$

Parameter	Symbol	μPD70108-5		μPD70108-8		μPD70108-10		Unit	Conditions
		Min	Max	Min	Max	Min	Max		
Small/Large Scale									
Clock cycle	t_{CYK}	200	500	125	500	100	500	ns	
Clock pulse width high	t_{KKH}	69		44		41		ns	$V_{KH} = 3.0\ V$
Clock pulse width low	t_{KKL}	90		60		49		ns	$V_{KL} = 1.5\ V$
Clock rise time	t_{KR}		10		8		5	ns	1.5 V to 3.0 V
Clock fall time	t_{KF}		10		7		5	ns	3.0 V to 1.5 V
READY inactive setup to CLK↓	t_{SRYLK}	−8		−8		−10		ns	
READY inactive hold after CLK↑	t_{HKRYH}	30		20		20		ns	
READY active setup to CLK↑	t_{SRYHK}	$t_{KKL} - 8$		$t_{KKL} - 8$		$t_{KKL} - 10$		ns	
READY active hold after CLK↑	t_{HKRYL}	30		20		20		ns	
Data setup time to CLK ↓	t_{SDK}	30		20		10		ns	
Data hold time after CLK ↓	t_{HKD}	10		10		10		ns	
NMI, INT, $\overline{POLL}$ setup time to CLK ↑	t_{SIK}	30		15		15		ns	
Input rise time (except CLK)	t_{IR}		20		20		20	ns	0.8 V to 2.2 V
Input fall time (except CLK)	t_{IF}		12		12		12	ns	2.2 V to 0.8 V
Output rise time	t_{OR}		20		20		20	ns	0.8 V to 2.2 V
Output fall time	t_{OF}		12		12		12	ns	2.2 V to 0.8 V
Small Scale									
Address delay time from CLK	t_{DKA}	10	90	10	60	10	48	ns	
Address hold time from CLK	t_{HKA}	10		10		10		ns	
PS delay time from CLK ↓	t_{DKP}	10	90	10	60	10	50	ns	
PS float delay time from CLK ↑	t_{FKP}	10	80	10	60	10	50	ns	
Address setup time to ASTB ↓	t_{SAST}	$t_{KKL} - 60$		$t_{KKL} - 30$		$t_{KKL} - 30$		ns	
Address float delay time from CLK ↓	t_{FKA}	t_{HKA}	80	t_{HKA}	60	t_{HKA}	50	ns	$C_L = 100\ pF$
ASTB ↑ delay time from CLK ↓	t_{DKSTH}		80		50		40	ns	
ASTB ↓ delay time from CLK ↑	t_{DKSTL}		85		55		45	ns	
ASTB width high	t_{STST}	$t_{KKL} - 20$		$t_{KKL} - 10$		$t_{KKL} - 10$		ns	
Address hold time from ASTB ↓	t_{HSTA}	$t_{KKH} - 10$		$t_{KKH} - 10$		$t_{KKH} - 10$		ns	

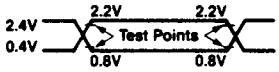
AC Characteristics (cont)

μPD70108-5, $T_A = -40°C$ to $+85°C$, $V_{DD} = +5\ V \pm 10\%$
μPD70108-8, μPD70108-10, $T_A = -10°C$ to $+70°C$, $V_{DD} = +5\ V \pm 5\%$

Parameter	Symbol	μPD70108-5 Min	μPD70108-5 Max	μPD70108-8 Min	μPD70108-8 Max	μPD70108-10 Min	μPD70108-10 Max	Unit	Conditions
Small Scale (cont)									
Control delay time from CLK	t_{DKCT}	10	110	10	65	10	55	ns	
Address float to $\overline{RD}$↓	t_{AFRL}	0		0		0		ns	
$\overline{RD}$ ↓ delay time from CLK ↓	t_{DKRL}	10	165	10	80	10	70	ns	
$\overline{RD}$ ↑ delay time from CLK ↓	t_{DKRH}	10	150	10	80	10	60	ns	
Address delay time from $\overline{RD}$ ↑	t_{DRHA}	$t_{CYK} - 45$		$t_{CYK} - 40$		$t_{CYK} - 35$		ns	
$\overline{RD}$ width low	t_{RR}	$2t_{CYK}-75$		$2t_{CYK}-50$		$2t_{CYK}-40$		ns	$C_L = 100$ pF
Data output delay time from CLK ↓	t_{DKD}	10	90	10	60	10	50	ns	
Data float delay time from CLK ↓	t_{FKD}	10	80	10	60	10	50	ns	
$\overline{WR}$ width low	t_{WW}	$2t_{CYK}-60$		$2t_{CYK}-40$		$2t_{CYK}-35$		ns	
HLDRQ setup time to CLK ↑	t_{SHQK}	35		20		20		ns	
HLDAK delay time from CLK ↓	t_{DKHA}	10	160	10	100	10	60	ns	
Large Scale									
Address delay time from CLK	t_{DKA}	10	90	10	60	10	48	ns	
Address hold time from CLK	t_{HKA}	10		10		10		ns	
PS delay time from CLK ↓	t_{DKP}	10	90	10	60	10	50	ns	
PS float delay time from CLK ↑	t_{FKP}	10	80	10	60	10	50	ns	
Address float delay time from CLK ↓	t_{FKA}	t_{HKA}	80	t_{HKA}	60	t_{HKA}	50	ns	
Address delay time from $\overline{RD}$ ↑	t_{DRHA}	$t_{CYK} - 45$		$t_{CYK} - 40$		$t_{CYK} - 35$		ns	
ASTB delay time from BS ↓	t_{DBST}		15		15		15	ns	
BS ↓ delay time from CLK ↑	t_{DKBL}	10	110	10	60	10	50	ns	
BS ↑ delay time from CLK ↓	t_{DKBH}	10	130	10	65	10	50	ns	
$\overline{RD}$ ↓ delay time from address float	t_{DAFRL}	0		0		0		ns	$C_L = 100$ pF
$\overline{RD}$ ↓ delay time from CLK ↓	t_{DKRL}	10	165	10	80	10	70	ns	
$\overline{RD}$ ↑ delay time from CLK ↓	t_{DKRH}	10	150	10	80	10	60	ns	
$\overline{RD}$ width low	t_{RR}	$2t_{CYK}-75$		$2t_{CYK}-50$		$2t_{CYK}-40$		ns	
Date output delay time from CLK ↓	t_{DKD}	10	90	10	60	10	50	ns	
Data float delay time from CLK ↑	t_{FKD}	10	80	10	60	10	50	ns	
$\overline{AK}$ delay time from CLK ↓	t_{DKAK}		70		50		40	ns	
$\overline{RQ}$ setup time to CLK ↑	t_{SRQK}	20		10		9		ns	
$\overline{RQ}$ hold time after CLK ↑	t_{HKRQ}	40		30		20		ns	

Timing Waveforms

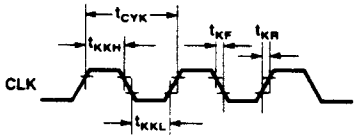
AC Test Input Waveform [Except CLK]



AC Output Test Points

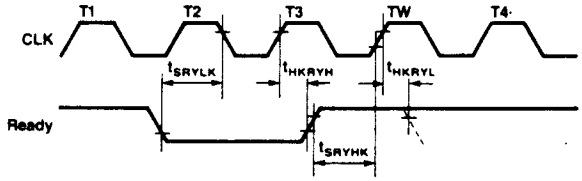
49-000238A

Clock Timing

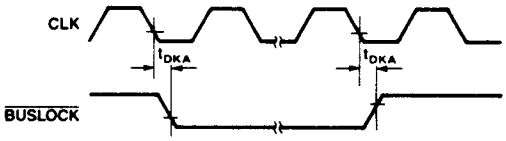


49-000239A

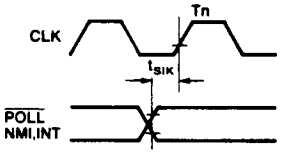
Wait [Ready] Timing



$\overline{\text{BUSLOCK}}$ Output Timing



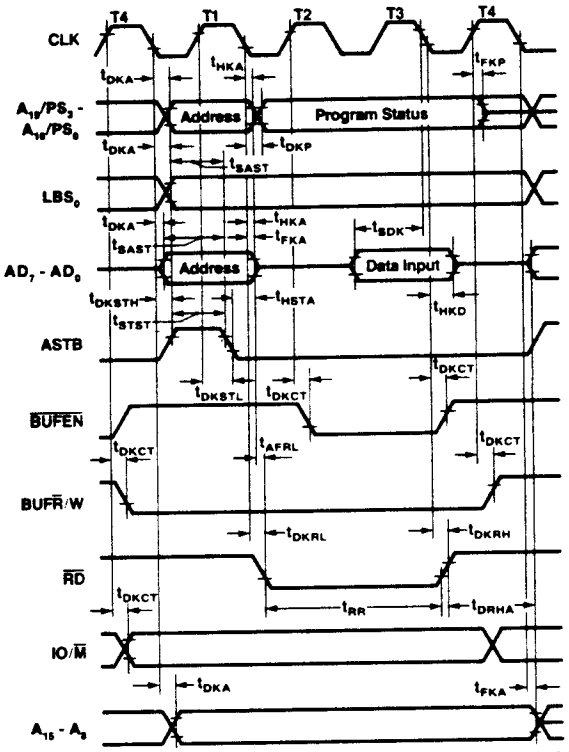
$\overline{\text{POLL}}$, NMI, INT Input Timing



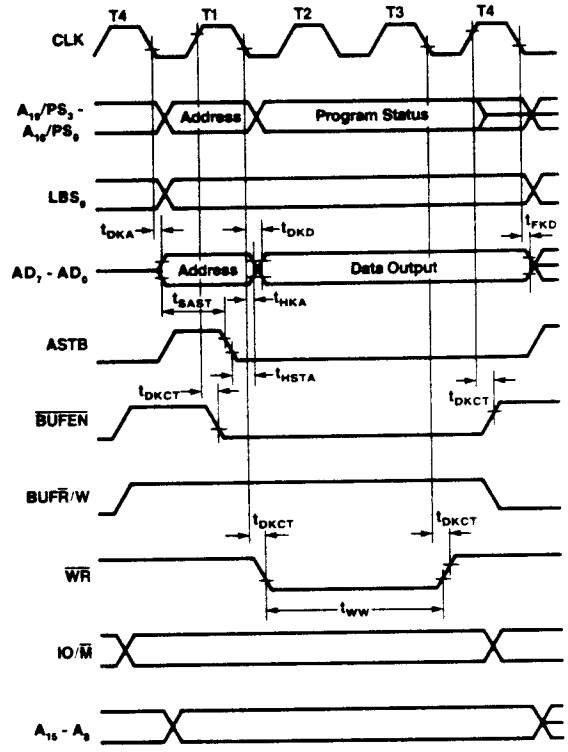
49-000240A

Timing Waveforms (cont)

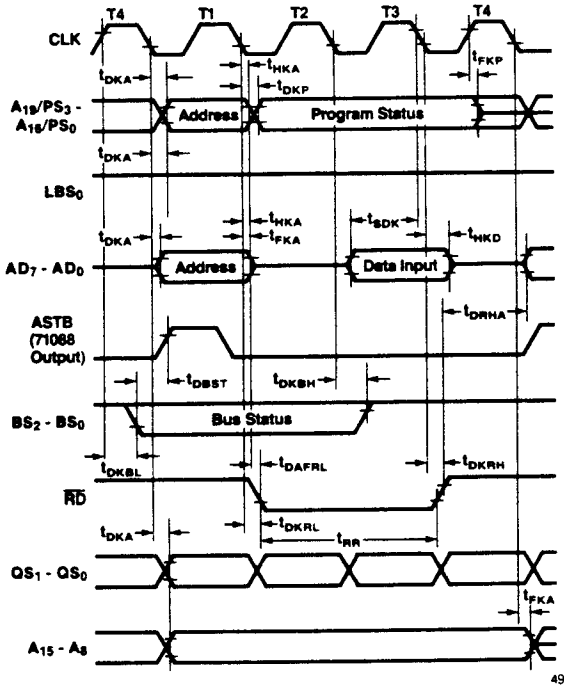
Read Timing [Small Scale]



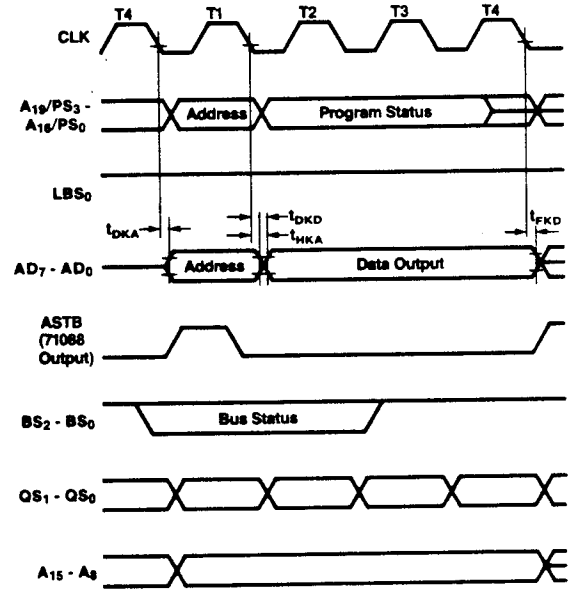
Write Timing [Small Scale]



Read Timing [Large Scale]

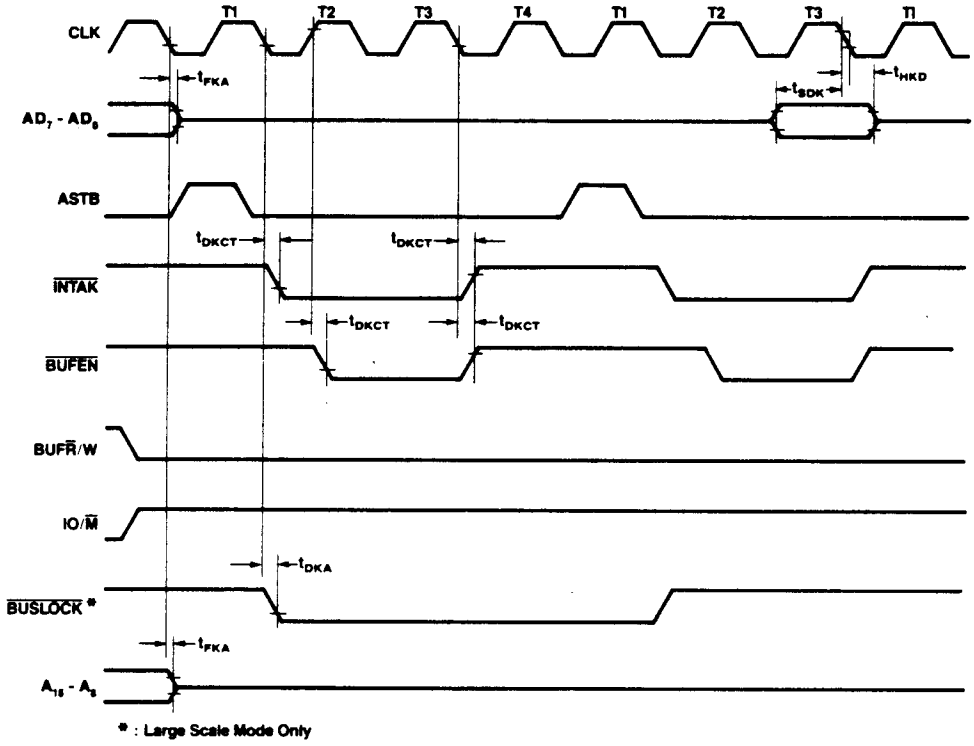


Write Timing [Large Scale]



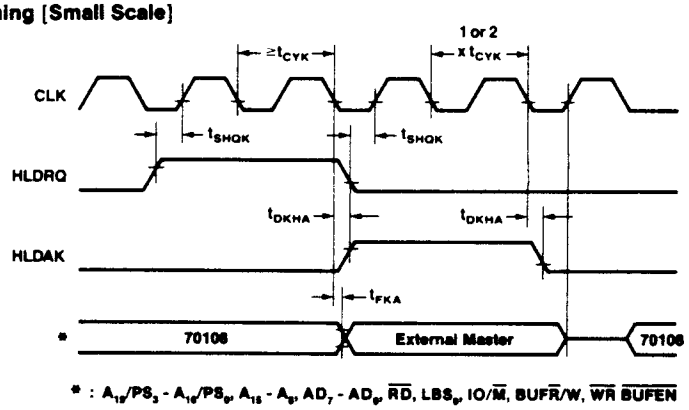
Timing Waveforms (cont)

Interrupt Acknowledge Timing



49:000245B

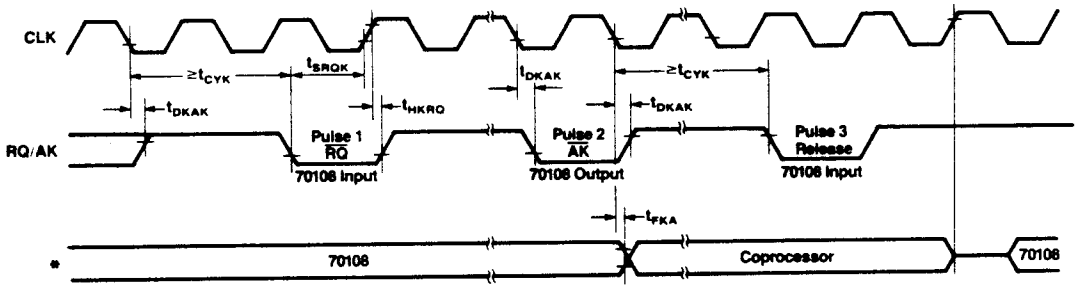
Hold Request/Acknowledge Timing [Small Scale]



49:000250B

Timing Waveforms (cont)

Bus Request/Acknowledge Timing [Large Scale]



* : A_{19}/PS_3 - A_{16}/PS_0, A_{15} - A_8, AD_7 - AD_0, BS_2 - BS_0, $\overline{RD}$, $\overline{BUSLOCK}$

49 0002-6B

Register Configuration

Program Counter [PC]

The program counter is a 16-bit binary counter that contains the segment offset address of the next instruction which the EXU is to execute.

The PC increments each time the microprogram fetches an instruction from the instruction queue. A new location value is loaded into the PC each time a branch, call, return, or break instruction is executed. At this time, the contents of the PC are the same as the Prefetch Pointer (PFP).

Prefetch Pointer [PFP]

The prefetch pointer (PFP) is a 16-bit binary counter which contains a segment offset which is used to calculate a program memory address that the bus control unit (BCU) uses to prefetch the next byte for the instruction queue. The contents of PFP are an offset from the PS (Program Segment) register.

The PFP is incremented each time the BCU prefetches an instruction from the program memory. A new location will be loaded into the PFP whenever a branch, call, return, or break instruction is executed. At that time the contents of the PFP will be the same as those of the PC (Program Counter).

Segment Registers [PS, SS, DS_0, and DS_1]

The memory addresses accessed by the μPD70108 are divided into 64K-byte logical segments. The starting (base) address of each segment is specified by a 16-bit segment register, and the offset from this starting address is specified by the contents of another register or by the effective address.

These are the four types of segment registers used.

Segment Register	Default Offset
PS (Program Segment)	PFP
SS (Stack Segment)	SP, effective address
DS_0 (Data Segment 0)	IX, effective address
DS_1 (Data Segment 1)	IY

General-Purpose Registers [AW, BW, CW, and DW]

There are four 16-bit general-purpose registers. Each one can be used as one 16-bit register or as two 8-bit registers by dividing them into their high and low bytes (AH, AL, BH, BL, CH, CL, DH, DL).

Each register is also used as a default register for processing specific instructions. The default assignments are:

AW: Word multiplication/division, word I/O, data conversion

AL: Byte multiplication/division, byte I/O, BCD rotation, data conversion, translation

AH: Byte multiplication/division

BW: Translation

CW: Loop control branch, repeat prefix

CL: Shift instructions, rototation instructions, BCD operations

DW: Word multiplication/division, indirect addressing I/O

Pointers [SP, BP] and Index Registers [IX, IY]

These registers serve as base pointers or index registers when accessing the memory using based addressing, indexed addressing, or based indexed addressing.

These registers can also be used for data transfer and arithmetic and logical operations in the same manner as the general-purpose registers. They cannot be used as 8-bit registers.

Also, each of these registers acts as a default register for specific operations. The default assignments are:

SP: Stack operations

IX: Block transfer (source), BCD string operations

IY: Block transfer (destination), BCD string operations

Program Status Word [PSW]

The program status word consists of the following six status and four control flags.

Status Flags

- V (Overflow)
- S (Sign)
- Z (Zero)
- AC (Auxiliary Carry)
- P (Parity)
- CY (Carry)

Control Flags

- MD (Mode)
- DIR (Direction)
- IE (Interrupt Enable)
- BRK (Break)

When the PSW is pushed on the stack, the word images of the various flags are as shown here.

PSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MD	1	1	1	V	DIR	IE	BRK	S	Z	0	AC	0	P	1	CY

The status flags are set and reset depending upon the result of each type of instruction executed.

Instructions are provided to set, reset, and complement the CY flag directly.

Other instructions set and reset the control flags and control the operation of the CPU.

High-Speed Execution of Instructions

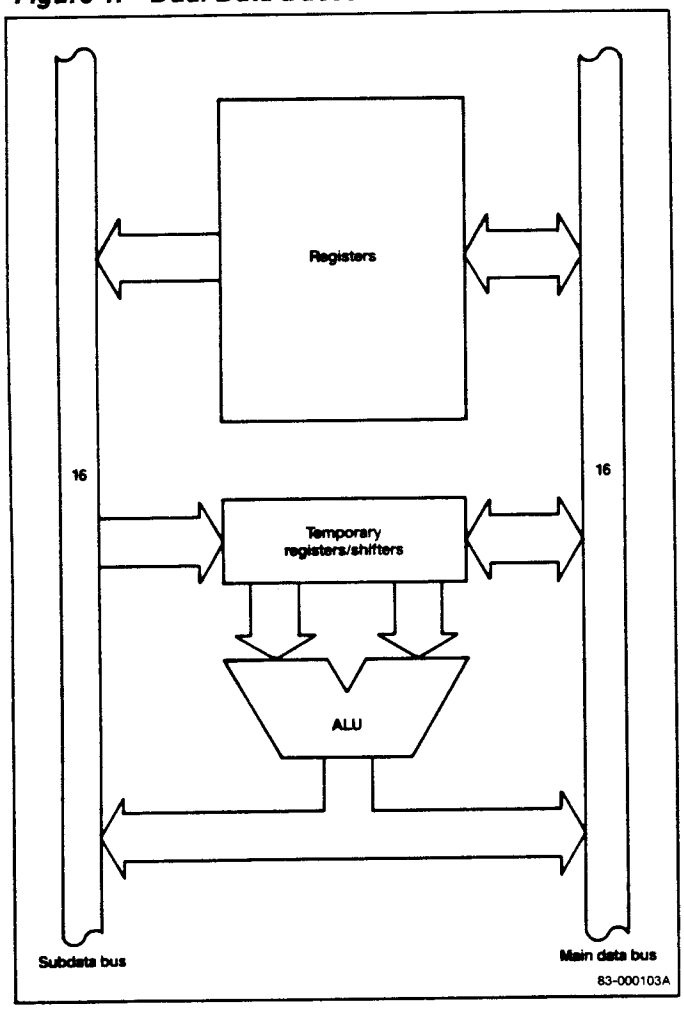
This section highlights the major architectural features that enhance the performance of the μPD70108.

- Dual data bus in EXU
- Effective address generator
- 16/32-bit temporary registers/shifters (TA, TB)
- 16-bit loop counter
- PC and PFP

Dual Data Bus Method

To reduce the number of processing steps for instruction execution, the dual data bus method has been adopted for the μPD70108 (figure 1). The two data buses (the main data bus and the subdata bus) are both 16 bits wide. For addition/subtraction and logical and comparison operations, processing time has been speeded up some 30% over single-bus systems.

Figure 1. Dual Data Buses



Example

ADD AW, BW ; AW ← AW + BW

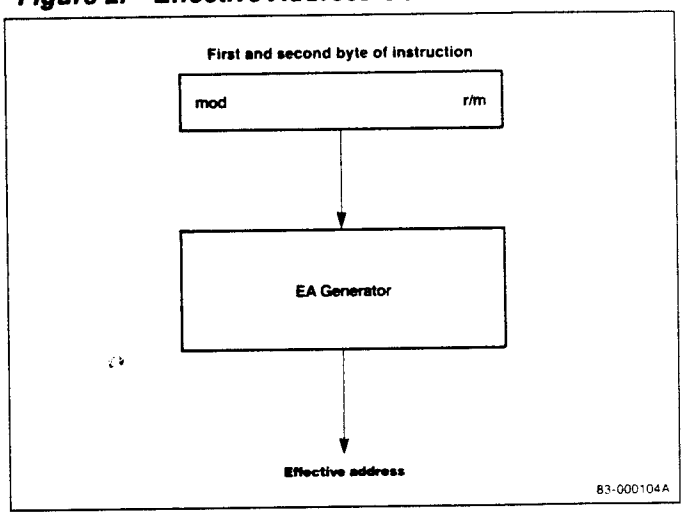
Single Bus	Dual Bus
Step 1 TA ← AW	TA ← AW, TB ← BW
Step 2 TB ← BW	AW ← TA + TB
Step 3 AW ← TA + TB	

Effective Address Generator

This circuit (figure 2) performs high-speed processing to calculate effective addresses for accessing memory.

Calculating an effective address by the microprogramming method normally requires 5 to 12 clock cycles. This circuit requires only two clock cycles for addresses to be generated for any addressing mode. Thus, processing is several times faster.

Figure 2. Effective Address Generator



16/32-Bit Temporary Registers/Shifters [TA, TB]

These 16-bit temporary registers/shifters (TA, TB) are provided for multiplication/division and shift/rotation instructions.

These circuits have decreased the execution time of multiplication/division instructions. In fact, these instructions can be executed about four times faster than with the microprogramming method.

TA + TB: 32-bit temporary register/shifter for multiplication and division instructions.

TB: 16-bit temporary register/shifter for shift/rotation instructions.

Loop Counter [LC]

This counter is used to count the number of loops for a primitive block transfer instruction controlled by a repeat prefix instruction and the number of shifts that will be performed for a multiple bit shift/rotation instruction.

The processing performed for a multiple bit rotation of a register is shown below. The average speed is approximately doubled over the microprogram method.

Example

RORC AW, CL ; CL = 5

Microprogram method	**LC method**
8 + (4 x 5) = 28 clocks	7 + 5 = 12 clocks

Program Counter and Prefetch Pointer [PC and PFP]

The μPD70108 microprocessor has a program counter, (PC) which addresses the program memory location of the instruction to be executed next, and a prefetch pointer(PFP), which addresses the program memory location to be accessed next. Both functions are provided in hardware. A time saving of several clocks is realized for branch, call, return, and break instruction execution, compared with microprocessors that have only one instruction pointer.

Enhanced Instructions

In addition to the μPD8088/86 instructions, the μPD70108 has the following enhanced instructions.

Instruction	Function
PUSH imm	Pushes immediate data onto stack
PUSH R	Pushes 8 general registers onto stack
POP R	Pops 8 general registers from stack
MUL imm	Executes 16-bit multiply of register or memory contents by immediate data
SHL imm8 SHR imm8 SHRA imm8 ROL imm8 ROR imm8 ROLC imm8 RORC imm8	Shifts/rotates register or memory by immediate value
CHKIND	Checks array index against designated boundaries
INM	Moves a string from an I/O port to memory
OUTM	Moves a string from memory to an I/O port
PREPARE	Allocates an area for a stack frame and copies previous frame pointers
DISPOSE	Frees the current stack frame on a procedure exit

Enhanced Stack Operation Instructions

PUSH imm

This instruction allows immediate data to be pushed onto the stack.

PUSH R/POP R

These instructions allow the contents of the eight general registers to be pushed onto or popped from the stack with a single instruction.

Enhanced Multiplication Instructions

MUL reg16, imm16/MUL mem16, imm16

These instructions allow the contents of a register or memory location to be 16-bit multiplied by immediate data.

Enhanced Shift and Rotate Instructions

SHL reg, imm8/SHR reg, imm8/SHRA reg, imm8

These instructions allow the contents of a register to be shifted by the number of bits defined by the immediate data.

ROL reg, imm8/ROR reg, imm8/ROLC reg, imm8/ RORC reg, imm8

These instructions allow the contents of a register to be rotated by the number of bits defined by the immediate data.

Check Array Boundary Instruction

CHKIND reg16, mem32

This instruction is used to verify that index values pointing to the elements of an array data structure are within the defined range. The lower limit of the array should be in memory location mem32, the upper limit in mem32 + 2. If the index value in reg16 is not between these limits when CHKIND is executed, a BRK 5 will occur. This causes a jump to the location in interrupt vector 5.

Block I/O Instructions

OUTM DW, src-block/INM dst-block, DW

These instructions are used to output or input a string to or from memory, when preceded by a repeat prefix.

Stack Frame Instructions

PREPARE imm16, imm8

This instruction is used to generate the stack frames required by block-structured languages, such as PASCAL and Ada. The stack frame consists of two areas. One area has a pointer that points to another frame which has variables that the current frame can access. The other is a local variable area for the current procedure.

DISPOSE

This instruction releases the last stack frame generated by the PREPARE instruction. It returns the stack and base pointers to the values they had before the PREPARE instruction was used to call a procedure.

Unique Instructions

In addition to the μPD8088/86 instructions and the enhanced instructions, the μPD70108 has the following unique instructions.

Instruction	Function
INS	Insert bit field
EXT	Extract bit field
ADD4S	Adds packed decimal strings
SUB4S	Subtracts one packed decimal string from another
CMP4S	Compares two packed decimal strings
ROL4	Rotates one BCD digit left through AL lower 4 bits
ROR4	Rotates one BCD digit right through AL lower 4 bits
TEST1	Tests a specified bit and sets/resets Z flag
NOT1	Inverts a specified bit
CLR1	Clears a specified bit
SET1	Sets a specified bit
REPC	Repeats next instruction until CY flag is cleared
REPNC	Repeats next instruction until CY flag is set
FPO2	Additional floating point processor call

Variable Length Bit Field Operation Instructions

This category has two instructions: INS (Insert Bit Field) and EXT (Extract Bit Field). These instructions are highly effective for computer graphics and high-level languages. They can, for example, be used for data structures such as packed arrays and record type data used in PASCAL.

INS reg8, reg8/INS reg8, imm4

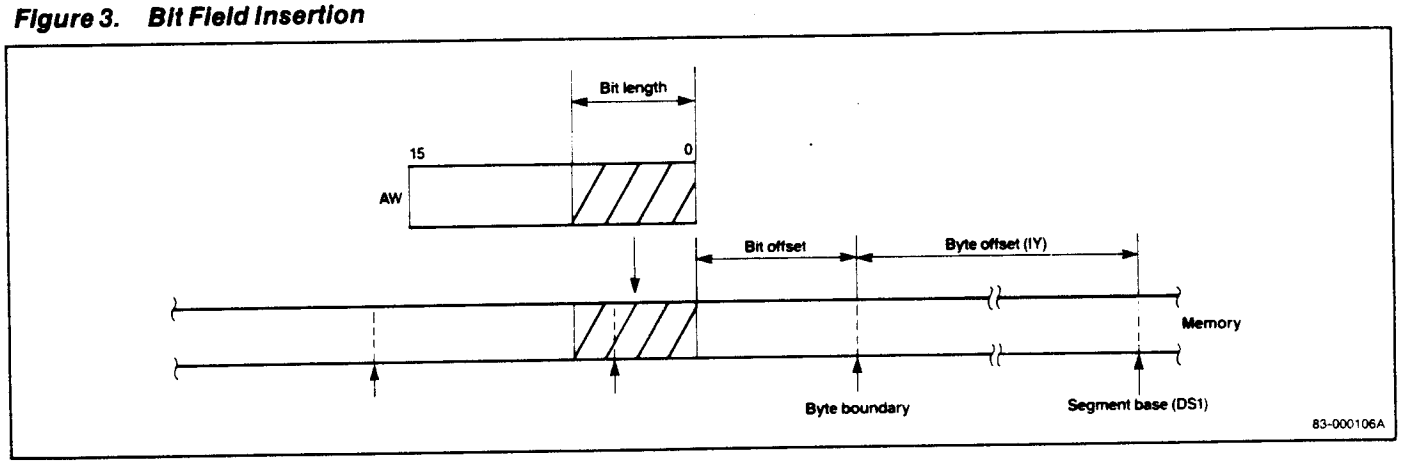
This instruction (figure 3) transfers low bits from the 16-bit AW register (the number of bits is specified by the second operand) to the memory location specified by the segment base (DS_1 register) plus the byte offset (IY register). The starting bit position within this byte is specified as an offset by the lower 4-bits of the first operand.

After each complete data transfer, the IY register and the register specified by the first operand are automatically updated to point to the next bit field.

Either immediate data or a register may specify the number of bits transferred (second operand). Because the maximum transferable bit length is 16-bits, only the lower 4-bits of the specified register (00H to 0FH) will be valid.

Bit field data may overlap the byte boundary of memory.

Figure 3. Bit Field Insertion



EXT reg8, reg8/EXT reg8, imm4

This instruction (figure 4) loads to the AW register the bit field data whose bit length is specified by the second operand of the instruction from the memory location that is specified by the DS0 segment register (segment base), the IX index register (byte offset), and the lower 4-bits of the first operand (bit offset).

After the transfer is complete, the IX register and the lower 4-bits of the first operand are automatically updated to point to the next bit field.

Either immediate data or a register may be specified for the second operand. Because the maximum transferrable bit length is 16 bits, however, only the lower 4-bits of the specified register (0H to 0FH) will be valid.

Bit field data may overlap the byte boundary of memory.

Packed BCD Operation Instructions

The instructions described here process packed BCD data either as strings (ADD4S, SUB4S, CMP4S) or byte-format operands (ROR4, ROL4). Packed BCD strings may be from 1 to 254 digits in length.

When the number of digits is even, the zero and carry flags will be set according to the result of the operation. When the number of digits is odd, the zero and carry flags may not be set correctly in this case, (CL = odd), the zero flag will not be set unless the upper 4 bits of the highest byte are all zero. The carry flag will not be set unless there is a carry out of the upper 4 bits of the highest byte. When CL is odd, the contents of the upper 4 bits of the highest byte of the result are undefined.

ADD4S

This instruction adds the packed BCD string addressed by the IX index register to the packed BCD string addressed by the IY index register, and stores the result in the string addressed by the IY register. The length of the string (number of BCD digits) is specified by the CL register, and the result of the operation will affect the overflow flag (V), the carry flag (CY), and zero flag (Z).

BCD string (IY, CL) ← BCD string (IY, CL) + BCD string (IX, CL)

SUB4S

This instruction subtracts the packed BCD string addressed by the IX index register from the packed BCD string addressed by the IY register, and stores the result in the string addressed by the IY register. The length of the string (number of BCD digits) is specified by the CL register, and the result of the operation will affect the overflow flag (V), the carry flag (CY), and zero flag (Z).

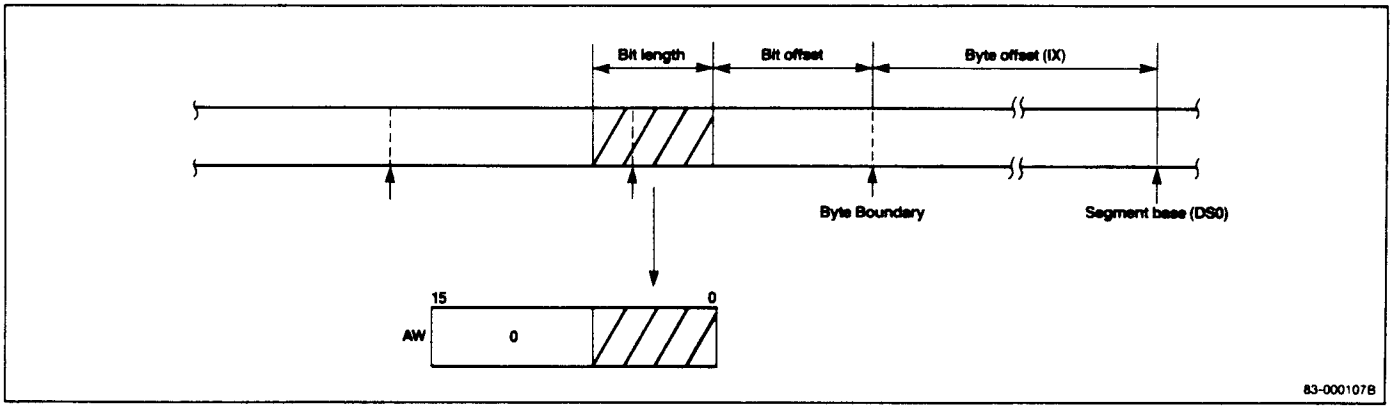
BCD string (IY, CL) ← BCD string (IY, CL) − BCD String (IX, CL)

CMP4S

This instruction performs the same operation as SUB4S except that the result is not stored and only the overflow (V), carry flags (CY) and zero flag (Z) are affected.

BCD string (IY, CL) − BCD string (IX, CL)

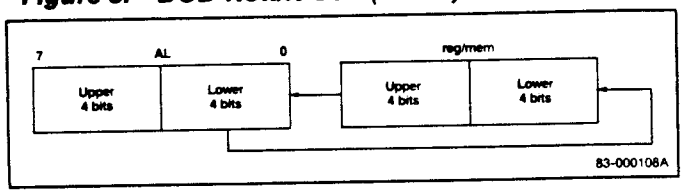
Figure 4. Bit Field Extraction



ROL4

This instruction (figure 5) treats the byte data of the register or memory directly specified by the instruction byte as BCD data and uses the lower 4-bits of the AL register (AL_L) to rotate that data one BCD digit to the left.

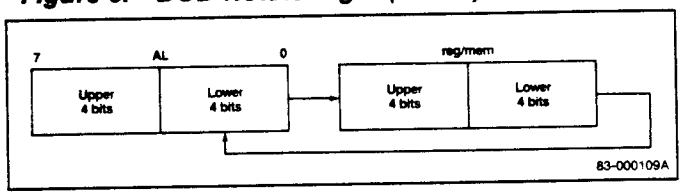
Figure 5. BCD Rotate Left (ROL4)



ROR4

This instruction (figure 6) treats the byte data of the register or memory directly specified by the instruction byte as BCD data and uses the lower 4-bits of the AL register (AL_L) to rotate that data one BCD digit to the right.

Figure 6. BCD Rotate Right (ROR4)



Bit Manipulation Instructions

TEST1

This instruction tests a specific bit in a register or memory location. If the bit is 1, the Z flag is reset to 0. If the bit is 0, the Z flag is set to 1.

NOT1

This instruction inverts a specific bit in a register or memory location.

CLR1

This instruction clears a specific bit in a register or memory location.

SET1

This instruction sets a specific bit in a register or memory location.

Repeat Prefix Instructions

REPC

This instruction causes the μPD70108 to repeat the following primitive block transfer instruction until the CY flag becomes cleared or the CW register becomes zero.

REPNC

This instruction causes the μPD70108 to repeat the following primitive block transfer instruction until the CY flag becomes set or the CW register is decremented to zero.

Floating Point Instruction

FPO2

This instruction is in addition to the μPD8088/86 floating point instruction, FPO1. These instructions are covered in a later section.

Mode Operation Instructions

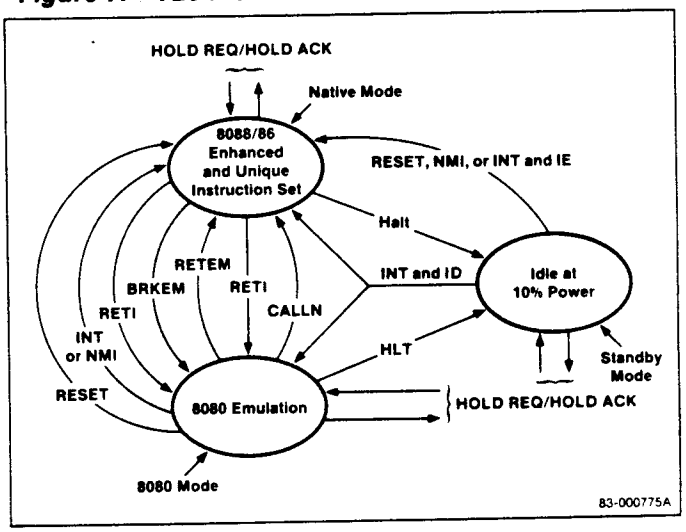
The μPD70108 has two operating modes (figure 7). One is the native mode which executes μPD8088/86, enhanced and unique instructions. The other is the 8080 emulation mode in which the instruction set of the μPD8080AF is emulated. A mode flag (MD) is provided to select between these two modes. Native mode is selected when MD is 1 and emulation mode when MD is 0. MD is set and reset, directly and indirectly, by executing the mode manipulation instructions.

Two instructions are provided to switch operation from the native mode to the emulation mode and back: BRKEM (Break for Emulation), and RETEM (Return from Emulation).

Two instructions are used to switch from the emulation mode to the native mode and back: CALLN (Call Native Routine), and RETI (Return from Interrupt).

The system will return from the 8080 emulation mode to the native mode when the RESET signal is present, or when an external interrupt (NMI or INT) is present.

Figure 7. V20 Modes



BRKEM imm8

This is the basic instruction used to start the 8080 emulation mode. This instruction operates exactly the same as the BRK instruction, except that BRKEM resets the mode flag (MD) to 0. PSW, PS, and PC are saved to the stack. MD is then reset and the interrupt vector specified by the operand imm8 of this command is loaded into PS and PC.

The instruction codes of the interrupt processing routine jumped to are then fetched. Then the CPU executes these codes as μPD8080AF instructions.

In 8080 emulation mode, registers and flags of the μPD8080AF are performed by the following registers and flags of the μPD70108.

	μPD8080AF	μPD70108
Registers:	A	AL
	B	CH
	C	CL
	D	DH
	E	DL
	H	BH
	L	BL
	SP	BP
	PC	PC
Flags:	C	CY
	Z	Z
	S	S
	P	P
	AC	AC

In the native mode, SP is used for the stack pointer. In the 8080 emulation mode this function is performed by BP.

This use of independent stack pointers allows independent stack areas to be secured for each mode and keeps the stack of one of the modes from being destroyed by an erroneous stack operation in the other mode.

The SP, IX, IY and AH registers and the four segment registers (PS, SS, DS_0, and DS_1) used in the native mode are not affected by operations in 8080 emulation mode.

In the 8080 emulation mode, the segment register for instructions is determined by the PS register (set automatically by the interrupt vector) and the segment register for data is the DS_0 register (set by the programmer immediately before the 8080 emulation mode is entered).

It is prohibited to nest BRKEM instructions.

RETEM [no operand]

When RETEM is executed in 8080 emulation mode (interpreted by the CPU as a μPD8080AF instruction), the CPU restores PS, PC, and PSW (as it would when returning from an interrupt processing routine), and returns to the native mode. At the same time, the contents of the mode flag (MD) which was saved to the stack by the BRKEM instruction, is restored to MD = 1. The CPU is set to the native mode.

CALLN imm8

This instruction makes it possible to call the native mode subroutines from the 8080 emulation mode. To return from subroutine to the emulation mode, the RETI instruction is used.

The processing performed when this instruction is executed in the 8080 emulation mode (it is interpreted by the CPU as μPD8080AF instruction), is similar to that performed when a BRK instruction is executed in the native mode. The imm8 operand specifies an interrupt vector type. The contents of PS, PC, and PSW are pushed on the stack and an MD flag value of 0 is saved. The mode flag is set to 1 and the interrupt vector specified by the operand is loaded into PS and PC.

RETI [no operand]

This is a general-purpose instruction used to return from interrupt routines entered by the BRK instruction or by an external interrupt in the native mode. When this instruction is executed at the end of a subroutine entered by the execution of the CALLN instruction, the operation that restores PS, PC, and PSW is exactly the same as the native mode execution. When PSW is restored, however, the 8080 emulation mode value of the mode flag (MD) is restored, the CPU is set in emulation mode, and all subsequent instructions are interpreted and executed as μPD8080AF instructions.

RETI is also used to return from an interrupt procedure initiated by an NMI or INT interrupt in the emulation mode.

Floating Point Operation Chip Instructions

FPO1 fp-op, mem/FPO2 fp-op, mem

These instructions are used for the external floating point processor. The floating point operation is passed to the floating point processor when the CPU fetches one of these instructions. From this point the CPU performs only the necessary auxiliary processing (effective address calculation, generation of physical addresses, and start-up of the memory read cycle).

The floating point processor always monitors the instructions fetched by the CPU. When it interprets one as an instruction to itself, it performs the appropriate processing. At this time, the floating point processor chip uses either the address alone or both the address and read data of the memory read cycle executed by the CPU. This difference in the data used depends on which of these instructions is executed.

Note: During the memory read cycle initiated by the CPU for FPO1 or FPO2 execution, the CPU does not accept any read data on the data bus from memory. Although the CPU generates the memory address, the data is used by the floating point processor.

Interrupt Operation

The interrupts used in the μPD70108 can be divided into two types: interrupts generated by external interrupt requests and interrupts generated by software processing. These are the classifications.

External Interrupts

(a) NMI input (nonmaskable)
(b) INT input (maskable)

Software Processing

As the result of instruction execution

— When a divide error occurs during execution of the DIV or DIVU instruction
— When a memory-boundary-over error is detected by the CHKIND instruction

Conditional break instruction

— When V = 1 during execution of the BRKV instruction

Unconditional break instructions

— 1-byte break instruction: BRK3
— 2-byte break instruction: BRK imm8

Flag processing

— When stack operations are used to set the BRK flag

8080 Emulation mode instructions

— BRKEM imm8
— CALLN imm8

Interrupt Vectors

Starting addresses for interrupt processing routines are either determined automatically by a single location of the interrupt vector table or selected each time interrupt processing is entered.

The interrupt vector table is shown in figure 8. The table uses 1K bytes of memory addresses 000H to 3FFH and can store starting address data for a maximum of 256 vectors (4 bytes per vector).

The corresponding interrupt sources for vectors 0 to 5 are predetermined and vectors 6 to 31 are reserved. These vectors consequently cannot be used for general applications.

The BRKEM instruction and CALLN instruction (in the emulation mode) and the INT input are available for general applications for vectors 32 to 255.

A single interrupt vector is made up of 4 bytes (figure 9). The 2 bytes in the low addresses of memory are loaded into PC as the offset, and the high 2 bytes are loaded into PS as the base address. The bytes are combined in reverse order. The lower-order bytes in the vector become the most significant bytes in the PC and PS, and the higher-order bytes become the least significant bytes.

Figure 8. Interrupt Vector Table

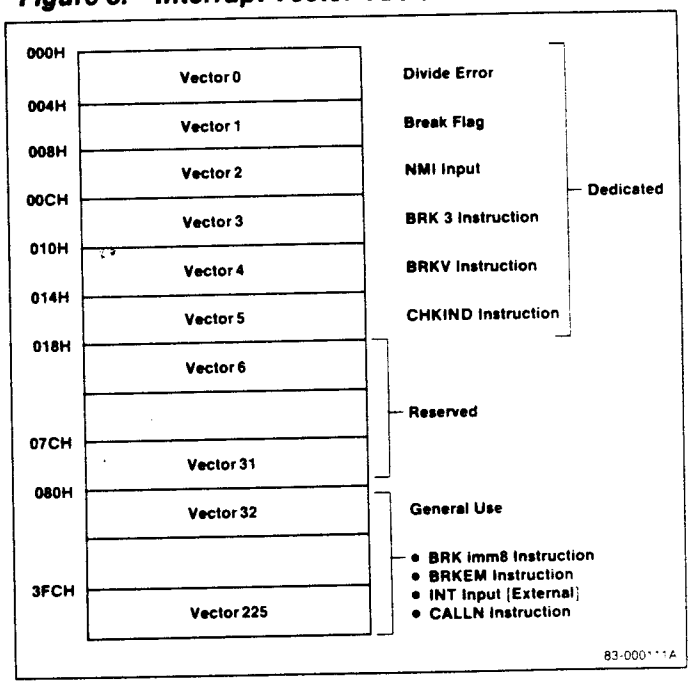
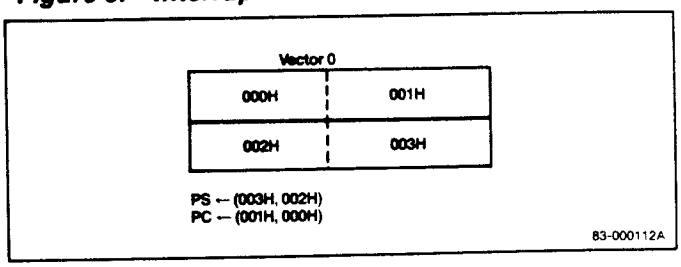


Figure 9. Interrupt Vector 0



Based on this format, the contents of each vector should be initialized at the beginning of the program.

The basic steps to jump to an interrupt processing routine are now shown.

(SP - 1, SP - 2) ← PSW
(SP - 3, SP - 4) ← PS
(SP - 5, SP - 6) ← PC
SP ← SP - 6
IE ← 0, BRK ← 0, MD ← 1
PS ← vector high bytes
PC ← vector low bytes

Standby Function

The μPD70108 has a standby mode to reduce power consumption during program wait states. This mode is set by the HALT instruction in both the native and the emulation mode.

In the standby mode, the internal clock is supplied only to those circuits related to functions required to release this mode and bus hold control functions. As a result, power consumption can be reduced to 1/10 the level of normal operation in either native or emulation mode.

The standby mode is released by inputting a RESET signal or an external interrupt (NMI, INT).

The bus hold function is effective during standby mode. The CPU returns to standby mode when the bus hold request is removed.

During standby mode, all control outputs are disabled and the addres/data bus will be at either high or low levels.

Instruction Set

The following tables briefly describe the μPD70108's instruction set.

- ☐ Operation and Operand Types - defines abbreviations used in the Instruction Set table.
- ☐ Flag Operations - defines the sybols used to describe flag operations.
- ☐ Memory Addressing - shows how mem and mod combinations specify memory addressing modes.
- ☐ Selection of 8- and 16-Bit Registers - shows how reg and W select a register when mod = 111.
- ☐ Selection of Segment Registers - shows how sreg selects a segment register.
- ☐ Instruction Set - shows the instruction mnemonics, their effect, their operation codes the number of bytes in the instruction, the number of clocks required for execution, and the effect on the μPD70108 flags.

Operation and Operand Types

Identifier	Description
reg	8- or 16-bit general-purpose register
reg8	8-bit general-purpose register
reg16	16-bit general-purpose register
dmem	8- or 16-bit direct memory location
mem	8- or 16-bit memory location
mem8	8-bit memory location
mem16	16-bit memory location
mem32	32-bit memory location
imm	Constant (0 to FFFFH)
imm16	Constant (0 to FFFFH)
imm8	Constant (0 to FFH)
imm4	Constant (0 to FH)
imm3	Constant (0 to 7)
acc	AW or AL register
sreg	Segment register
src-table	Name of 256-byte translation table
src-block	Name of block addressed by the IX register
dst-block	Name of block addressed by the IY register
near-proc	Procedure within the current program segment
far-proc	Procedure located in another program segment
near-label	Label in the current program segment
short-label	Label between −128 and +127 bytes from the end of instruction
far-label	Label in another program segment
memptr16	Word containing the offset of the memory location within the current program segment to which control is to be transferred
memptr32	Double word containing the offset and segment base address of the memory location to which control is to be transferred
regptr16	16-bit register containing the offset of the memory location within the program segment to which control is to be transferred
pop-value	Number of bytes of the stack to be discarded (0 to 64K bytes, usually even addresses)
fp-op	Immediate data to identify the instruction code of the external floating point operation

Operation and Operand Types (cont)

Identifier	Description
R	Register set
W	Word/byte field (0 to 1)
reg	Register field (000 to 111)
mem	Memory field (000 to 111)
mod	Mode field (00 to 10)
S:W	When S:W = 01 or 11, data = 16 bits. At all other times, data = 8 bits.
X, XXX, YYY, ZZZ	Data to identify the instruction code of the external floating point arithmetic chip
AW	Accumulator (16 bits)
AH	Accumulator (high byte)
AL	Accumulator (low byte)
BW	BW register (16 bits)
CW	CW register (16 bits)
CL	CW register (low byte)
DW	DW register (16 bits)
SP	Stack pointer (16 bits)
PC	Program counter (16 bits)
PSW	Program status word (16 bits)
IX	Index register (source) (16 bits)
IY	Index register (destination) (16 bits)
PS	Program segment register (16 bits)
SS	Stack segment register (16 bits)
DS_0	Data segment 0 register (16 bits)
DS_1	Data segment 1 register (16 bits)
AC	Auxiliary carry flag
CY	Carry flag
P	Parity flag
S	Sign flag
Z	Zero flag
DIR	Direction flag
IE	Interrupt enable flag
V	Overflow flag
BRK	Break flag
MD	Mode flag
(...)	Values in parentheses are memory contents
disp	Displacement (8 or 16 bits)
ext-disp8	16-bit displacement (sign-extension byte + 8-bit displacement)
temp	Temporary register (8/16/32 bits)

Operation and Operand Types (cont)

Identifier	Description
tmpcy	Temporary carry flag (1 bit)
seg	Immediate segment data (16 bits)
offset	Immediate offset data (16 bits)
←	Transfer direction
+	Addition
−	Subtraction
x	Multiplication
÷	Division
%	Modulo
AND	Logical product
OR	Logical sum
XOR	Exclusive logical sum
XXH	Two-digit hexadecimal value
XXXXH	Four-digit hexadecimal value

Flag Operations

Identifier	Description
(blank)	No change
0	Cleared to 0
1	Set to 1
X	Set or cleared according to the result
U	Undefined
R	Value saved earlier is restored

Memory Addressing

	mod		
mem	00	01	10
000	BW + IX	BW + IX + disp8	BW + IX + disp16
001	BW + IY	BW + IY + disp8	BW + IY + disp16
010	BP + IX	BP + IX + disp8	BP + IX + disp16
011	BP + IY	BP + IY + disp8	BP + IY + disp16
100	IX	IX + disp8	IX + disp16
101	IY	IY + disp8	IY + disp16
110	Direct address	BP + disp8	BP + disp16
111	BW	BW + disp8	BW + disp16

Selection of 8- and 16-Bit Registers (mod 11)

reg	W = 0	W = 1
000	AL	AW
001	CL	CW
010	DL	DW
011	BL	BW
100	AH	SP
101	CH	BP
110	DH	IX
111	BH	IY

Selection of Segment Registers

sreg	
00	DS_1
01	PS
10	SS
11	DS_0

The table on the following pages shows the instruction set.

At "No. of Clocks," for instructions referencing memory operands, the left side of the slash (/) is the number of clocks for byte operands and the right side is for word operands. For conditional control transfer instructions, the left side of the slash (/) is the number of clocks if a control transfer takes place. The right side is the number of clocks when no control transfer or branch occurs. Some instructions show a range of clock times, separated by a hyphen. The execution time of these instructions varies from the minimum value to the maximum, depending on the operands involved.

"No. of Clocks" includes these times:

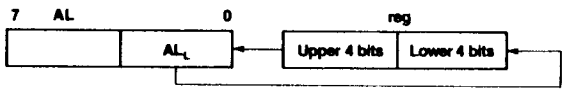
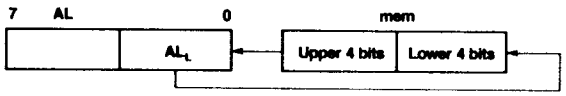
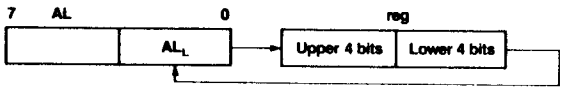
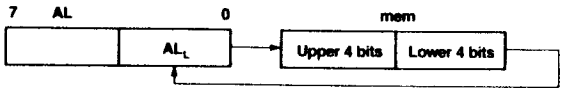
- Decoding
- Effective address generation
- Operand fetch
- Execution

It assumes that the instruction bytes have been pre-fetched.

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	Flags AC	CY	V	P	S	Z
		Data Transfer Instructions										
MOV	reg, reg	reg ← reg	1 0 0 0 1 0 1 W	1 1 reg reg	2	2						
	mem, reg	(mem) ← reg	1 0 0 0 1 0 0 W	mod reg mem	9/13	2-4						
	reg, mem	reg ← (mem)	1 0 0 0 1 0 1 W	mod reg mem	11/15	2-4						
	mem, imm	(mem) ← imm	1 1 0 0 0 1 1 W	mod 0 0 0 mem	11/15	3-6						
	reg, imm	reg ← imm	1 0 1 1 W reg		4	2-3						
	acc, dmem	When W = 0 AL ← (dmem) When W = 1 AH ← (dmem + 1), AL ← (dmem)	1 0 1 0 0 0 0 W		10/14	3						
	dmen, acc	When W = 0 (dmem) ← AL When W = 1 (dmem + 1) ← AH, (dmem) ← AL	1 0 1 0 0 0 1 W		9/13	3						
	sreg, reg16	sreg ← reg16 sreg : SS, DS0, DS1	1 0 0 0 1 1 1 0	1 1 0 sreg reg	2	2						
	sreg, mem16	sreg ← (mem16) sreg : SS, DS0, DS1	1 0 0 0 1 1 1 0	mod 0 sreg mem	11/15	2-4						
	reg16, sreg	reg16 ← sreg	1 0 0 0 1 1 0 0	1 1 0 sreg reg	2	2						
	mem16, sreg	(mem16) ← sreg	1 0 0 0 1 1 0 0	mod 0 sreg mem	10/14	2-4						
	DS0, reg16, mem32	reg16 ← (mem32) DS0 ← (mem32 + 2)	1 1 0 0 0 1 0 1	mod reg mem	18/26	2-4						
	DS1, reg16, mem32	reg16 ← (mem32) DS1 ← (mem32 + 2)	1 1 0 0 0 1 0 0	mod reg mem	18/26	2-4						
	AH, PSW	AH ← S, Z, x, AC, x, P, x, CY	1 0 0 1 1 1 1 1		2	1	x	x		x	x	x
	PSW, AH	S, Z, x, AC, x, P, x, CY ← AH	1 0 0 1 1 1 1 0		3	1	x	x		x	x	x
LDEA	reg16, mem16	reg16 ← mem16	1 0 0 0 1 1 0 1	mod reg mem	4	2-4						
TRANS	src-table	AL ← (BW + AL)	1 1 0 1 0 1 1 1		9	1						
XCH	reg, reg	reg ⟷ reg	1 0 0 0 0 1 1 W	1 1 reg reg	3	2						
	mem, reg or reg, mem	(mem) ⟷ reg	1 0 0 0 0 1 1 W	mod reg mem	16/26	2-4						
	AW, reg16 or reg16, AW	AW ⟷ reg16	1 0 0 1 0 reg		3	1						
		Repeat Prefixes										
REPC		While CW ≠ 0, the next byte of the primitive block transfer instruction is executed and CW is decremented (− 1). If there is a waiting interrupt, it is processed. When CY ≠ 1, exit the loop.	0 1 1 0 0 1 0 1		2	1						
REPNC		While CW ≠ 0, the next byte of the primitive block transfer instruction is executed and CW is decremented (− 1). If there is a waiting interrupt, it is processed. When CY ≠ 0, exit the loop.	0 1 1 0 0 1 0 0		2	1						

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	Flags AC	CY	V	P	S	Z
		Repeat Prefixes (cont)										
REP REPE REPZ		While CW ≠ 0, the next byte of the primitive block transfer instruction is executed and CW is decremented (− 1). If there is a waiting interrupt, it is processed. If the primitive block transfer instruction is CMPBK or CMPM and Z ≠ 1, exit the loop.	1 1 1 1 0 0 1 1		2	1						
REPNE REPNZ		While CW ≠ 0, the next byte of the primitive block transfer instruction is executed and CW is decremented (− 1). If there is a waiting interrupt, it is processed. If the primitive block transfer instruction is CMPBK or CMPM and Z ≠ 0, exit the loop.	1 1 1 1 0 0 1 0		2	1						
		Primitive Block Transfer Instructions										
MOVBK	dst-block, src-block	When W = 0 (IY) ← (IX) DIR = 0: IX ← IX + 1, IY ← IY + 1 DIR = 1: IX ← IX − 1, IY ← IY − 1 When W = 1 (IY + 1, IY) ← (IX + 1, IX) DIR = 0: IX ← IX + 2, IY ← IY + 2 DIR = 1: IX ← IX − 2, IY ← IY − 2	1 0 1 0 0 1 0 W		11 + 8n 11 + 16n	1						
CMPBK	src-block, dst-block	When W = 0 (IX) − (IY) DIR = 0: IX ← IX + 1, IY ← IY + 1 DIR = 1: IX ← IX − 1, IY ← IY − 1 When W = 1 (IX + 1, IX) − (IY + 1, IY) DIR = 0: IX ← IX + 2, IY ← IY + 2 DIR = 1: IX ← IX − 2, IY ← IY − 2	1 0 1 0 0 1 1 W		7 + 14n 7 + 22n	1	x	x	x	x	x	x
CMPM	dst-block	When W = 0 AL − (IY) DIR = 0: IY ← IY + 1; DIR = 1: IY ← IY − 1 When W = 1 AW − (IY + 1, IY) DIR = 0: IY ← IY + 2; DIR = 1: IY ← IY − 2	1 0 1 0 1 1 1 W		7 + 10n 7 + 14n	1	x	x	x	x	x	x
LDM	src-block	When W = 0 AL ← (IX) DIR = 0: IX ← IX + 1; DIR = 1: IX ← IX − 1 When W = 1 AW ← (IX + 1, IX) DIR = 0: IX ← IX + 2; DIR = 1: IX ← IX − 2	1 0 1 0 1 1 0 W		7 + 9n 7 + 13n	1						
STM	dst-block	When W = 0 (IY) ← AL DIR = 0: IY ← IY + 1; DIR = 1: IY ← IY − 1 When W = 1 (IY + 1, IY) ← AW DIR = 0: IY ← IY + 2; DIR = 1: IY ← IY − 2	1 0 1 0 1 0 1 W n: number of transfers		7 + 4n 7 + 8n	1						
		Bit Field Transfer Instructions										
INS	reg8, reg8	16-Bit field ← AW	0 0 0 0 1 1 1 1 1 1 reg reg	0 0 1 1 0 0 0 1	35-133	3						
	reg8, imm4	16-Bit field ← AW	0 0 0 0 1 1 1 1 1 1 0 0 0 reg	0 0 1 1 1 0 0 1	35-133	4						

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	Flags AC	CY	V	P	S	Z
		Bit Field Transfer Instructions (cont)										
EXT	reg8, reg8	AW ← 16-Bit field	0 0 0 0 1 1 1 1 1 1 reg reg	0 0 1 1 0 0 1 1	34-59	3						
	reg8, imm4	AW ← 16-Bit field	0 0 0 0 1 1 1 1 1 1 0 0 0 reg	0 0 1 1 1 0 1 1	34-59	4						
		I/O Instructions										
IN	acc, imm8	When W = 0 AL ← (imm8) When W = 1 AH ← (imm8 + 1), AL ← (imm8)	1 1 1 0 0 1 0 W		9/13	2						
	acc, DW	When W = 0 AL ← (DW) When W = 1 AH ← (DW + 1), AL ← (DW)	1 1 1 0 1 1 0 W		8/12	1						
OUT	imm8, acc	When W = 0 (imm8) ← AL When W = 1 (imm8 + 1) ← AH, (imm8) ← AL	1 1 1 0 0 1 1 W		8/12	2						
	DW, acc	When W = 0 (DW) ← AL When W = 1 (DW + 1) ← AH, (DW) ← AL	1 1 1 0 1 1 1 W		8/12	1						
		Primitive I/O Instructions										
INM	dst-block, DW	When W = 0 (IY) ← (DW) DIR = 0: IY ← IY + 1; DIR = 1: IY ← IY − 1 When W = 1 (IY + 1, IY) ← (DW + 1, DW) DIR = 0: IY ← IY + 2; DIR = 1: IY ← IY − 2	0 1 1 0 1 1 0 W		9 + 8n 9 + 16n	1						
OUTM	DW, src-block	When W = 0 (DW) ← (IX) DIR = 0: IX ← IX + 1; DIR = 1: IX ← IX − 1 When W = 1 (DW + 1, DW) ← (IX + 1, IX) DIR = 0: IX ← IX + 2; DIR = 1: IX ← IX − 2	0 1 1 0 1 1 1 W n: number of transfers		9 + 8n 9 + 16n	1						
		Addition/Subtraction Instructions										
ADD	reg, reg	reg ← reg + reg	0 0 0 0 0 0 1 W	1 1 reg reg	2	2	x	x	x	x	x	x
	mem, reg	(mem) ← (mem) + reg	0 0 0 0 0 0 0 W	mod reg mem	16/24	2-4	x	x	x	x	x	x
	reg, mem	reg ← reg + (mem)	0 0 0 0 0 0 1 W	mod reg mem	11/15	2-4	x	x	x	x	x	x
	reg, imm	reg ← reg + imm	1 0 0 0 0 0 S W	1 1 0 0 0 reg	4	3-4	x	x	x	x	x	x
	mem, imm	(mem) ← (mem) + imm	1 0 0 0 0 0 S W	mod 0 0 0 mem	18/26	3-6	x	x	x	x	x	x
	acc, imm	When W = 0 AL ← AL + imm When W = 1 AW ← AW + imm	0 0 0 0 0 1 0 W		4	2-3	x	x	x	x	x	x
ADDC	reg, reg	reg ← reg + reg + CY	0 0 0 1 0 0 1 W	1 1 reg reg	2	2	x	x	x	x	x	x
	mem, reg	(mem) ← (mem) + reg + CY	0 0 0 1 0 0 0 W	mod reg mem	16/24	2-4	x	x	x	x	x	x
	reg, mem	reg ← reg + (mem) + CY	0 0 0 1 0 0 1 W	mod reg mem	11/15	2-4	x	x	x	x	x	x
	reg, imm	reg ← reg + imm + CY	1 0 0 0 0 0 S W	1 1 0 1 0 reg	4	3-4	x	x	x	x	x	x
	mem, imm	(mem) ← (mem) + imm + CY	1 0 0 0 0 0 S W	mod 0 1 0 mem	18/26	3-6	x	x	x	x	x	x

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	Flags AC	CY	V	P	S	Z
		Addition/Subtraction Instructions (cont)										
ADDC	acc, imm	When W = 0 AL ← AL + imm + CY When W = 1 AW ← AW + imm + CY	0 0 0 1 0 1 0 W		4	2-3	x	x	x	x	x	x
SUB	reg, reg	reg ← reg − reg	0 0 1 0 1 0 1 W	1 1 reg reg	2	2	x	x	x	x	x	x
	mem, reg	(mem) ← (mem) − reg	0 0 1 0 1 0 0 W	mod reg mem	16/24	2-4	x	x	x	x	x	x
	reg, mem	reg ← reg − (mem)	0 0 1 0 1 0 1 W	mod reg mem	11/15	2-4	x	x	x	x	x	x
	reg, imm	reg ← reg − imm	1 0 0 0 0 0 S W	1 1 1 0 1 reg	4	3-4	x	x	x	x	x	x
	mem, imm	(mem) ← (mem) − imm	1 0 0 0 0 0 S W	mod 1 0 1 mem	18/26	3-6	x	x	x	x	x	x
	acc, imm	When W = 0 AL ← AL − imm When W = 1 AW ← AW − imm	0 0 1 0 1 1 0 W		4	2-3	x	x	x	x	x	x
SUBC	reg, reg	reg ← reg − reg − CY	0 0 0 1 1 0 1 W	1 1 reg reg	2	2	x	x	x	x	x	x
	mem, reg	(mem) ← (mem) − reg − CY	0 0 0 1 1 0 0 W	mod reg mem	16/24	2-4	x	x	x	x	x	x
	reg, mem	reg ← reg − (mem) − CY	0 0 0 1 1 0 1 W	mod reg mem	11/15	2-4	x	x	x	x	x	
	reg, imm	reg ← reg − imm − CY	1 0 0 0 0 0 S W	1 1 0 1 1 reg	4	3-4	x	x	x	x	x	x
	mem, imm	(mem) ← (mem) − imm − CY	1 0 0 0 0 0 S W	mod 0 1 1 mem	18/26	3-6	x	x	x	x	x	x
	acc, imm	When W = 0 AL ← AL − imm − CY When W = 1 AW ← AW − imm − CY	0 0 0 1 1 1 0 W		4	2-3	x	x	x	x	x	x
		BCD Operation Instructions										
ADD4S		dst BCD string ← dst BCD string + src BCD string	0 0 0 0 1 1 1 1	0 0 1 0 0 0 0 0	7 + 19n	2	u	x	u	u	u	x
SUB4S		dst BCD string ← dst BCD string − src BCD string	0 0 0 0 1 1 1 1	0 0 1 0 0 0 1 0	7 + 19n	2	u	x	u	u	u	x
CMP4S		dst BCD string − src BCD string	0 0 0 0 1 1 1 1	0 0 1 0 0 1 1 0	7 + 19n	2	u	x	u	u	u	x
			n: number of BCD digits divided by 2									
ROL4	reg8		0 0 0 0 1 1 1 1 1 1 0 0 0 reg	0 0 1 0 1 0 0 0	25	3						
	mem8		0 0 0 0 1 1 1 1 mod 0 0 0 mem	0 0 1 0 1 0 0 0	28	3-5						
ROR4	reg8		0 0 0 0 1 1 1 1 1 1 0 0 0 reg	0 0 1 0 1 0 1 0	29	3						
	mem8		0 0 0 0 1 1 1 1 mod 0 0 0 mem	0 0 1 0 1 0 1 0	33	3-5						

NEC

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	Flags AC	CY	V	P	S	Z
		Increment/Decrement Instructions (cont)										
INC	reg8	reg8 ← reg8 + 1	1 1 1 1 1 1 1 0	1 1 0 0 0 reg	2	2	x		x	x	x	x
	mem	(mem) ← (mem) + 1	1 1 1 1 1 1 1 W	mod 0 0 0 mem	16/24	2-4	x		x	x	x	x
	reg16	reg16 ← reg16 + 1	0 1 0 0 0 reg		2	1	x		x	x	x	x
DEC	reg8	reg8 ← reg8 − 1	1 1 1 1 1 1 1 0	1 1 0 0 1 reg	2	2	x		x	x	x	x
	mem	(mem) ← (mem) − 1	1 1 1 1 1 1 1 W	mod 0 0 1 mem	16/24	2-4	x		x	x	x	x
	reg16	reg16 ← reg16 − 1	0 1 0 0 1 reg		2	1	x		x	x	x	x
		Multiplication Instructions										
MULU	reg8	AW ← AL x reg8 AH = 0: CY ← 0, V ← 0 AH ≠ 0: CY ← 1, V ← 1	1 1 1 1 0 1 1 0	1 1 1 0 0 reg	21-22	2	u	x	x	u	u	u
	mem8	AW ← AL x (mem8) AH = 0: CY ← 0, V ← 0 AH ≠ 0: CY ← 1, V ← 1	1 1 1 1 0 1 1 0	mod 1 0 0 mem	27-28	2-4	u	x	x	u	u	u
	reg16	DW, AW ← AW x reg16 DW = 0: CY ← 0, V ← 0 DW ≠ 0: CY ← 1, V ← 1	1 1 1 1 0 1 1 1	1 1 1 0 0 reg	29-30	2	u	x	x	u	u	u
	mem16	DW, AW ← AW x (mem16) DW = 0: CY ← 0, V ← 0 DW ≠ 0: CY ← 1, V ← 1	1 1 1 1 0 1 1 1	mod 1 0 0 mem	39-40	2-4	u	x	x	u	u	u
MUL	reg8	AW ← AL x reg8 AH = AL sign expansion: CY ← 0, V ← 0 AH ≠ AL sign expansion: CY ← 1, V ← 1	1 1 1 1 0 1 1 0	1 1 1 0 1 reg	33-39	2	u	x	x	u	u	u
	mem8	AW ← AL x (mem8) AH = AL sign expansion: CY ← 0, V ← 0 AH ≠ AL sign expansion: CY ← 1, V ← 1	1 1 1 1 0 1 1 0	mod 1 0 1 mem	39-45	2-4	u	x	x	u	u	u
	reg16	DW, AW ← AW x reg16 DW = AW sign expansion: CY ← 0, V ← 0 DW ≠ AW sign expansion: CY ← 1, V ← 1	1 1 1 1 0 1 1 1	1 1 1 0 1 reg	41-47	2	u	x	x	u	u	u
	mem16	DW, AW ← AW x (mem16) DW = AW sign expansion: CY ← 0, V ← 0 DW ≠ AW sign expansion: CY ← 1, V ← 1	1 1 1 1 0 1 1 1	mod 1 0 1 mem	51-57	2-4	u	x	x	u	u	u
	reg16, (reg16,) imm8	reg16 ← reg16 x imm8 Product ≤ 16 bits: CY ← 0, V ← 0 Product > 16 bits: CY ← 1, V ← 1	0 1 1 0 1 0 1 1	1 1 reg reg	28-34	3	u	x	x	u	u	u
	reg16, mem16, imm8	reg16 ← (mem16) x imm8 Product ≤ 16 bits: CY ← 0, V ← 0 Product > 16 bits: CY ← 1, V ← 1	0 1 1 0 1 0 1 1	mod reg mem	38-44	3-5	u	x	x	u	u	u

Mnemonic	Operand	Operation	Operation Code 7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	No. of Clocks	No. of Bytes	Flags AC	CY	V	P	S	Z
		Multiplication Instructions (cont)																								
MUL	reg16, (reg16,) imm16	reg16 ← reg16 x imm16 Product ≤ 16 bits: CY ← 0, V ← 0 Product > 16 bits: CY ← 1, V ← 1	0	1	1	0	1	0	0	1	1	1		reg			reg		36-42	4	u	x	x	u	u	u
	reg16, mem16, imm16	reg16 ← (mem16) x imm16 Product ≤ 16 bits: CY ← 0, V ← 0 Product > 16 bits: CY ← 1, V ← 1	0	1	1	0	1	0	0	1	mod			reg			mem		46-52	4-6	u	x	x	u	u	u
		Unsigned Division Instructions																								
DIVU	reg8	temp ← AW When temp ÷ reg8 > FFH (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % reg8, AL ← temp ÷ reg8	1	1	1	1	0	1	1	0	1	1	1	1	0		reg		19	2	u	u	u	u	u	u
	mem8	temp ← AW When temp ÷ (mem8) > FFH (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % (mem8), AL ← temp ÷ (mem8)	1	1	1	1	0	1	1	0	mod		1	1	0		mem		25	2-4	u	u	u	u	u	u
	reg16	temp ← AW When temp ÷ reg16 > FFFFH (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % reg16, AL ← temp ÷ reg16	1	1	1	1	0	1	1	1	1	1	1	1	0		reg		25	2	u	u	u	u	u	u
	mem16	temp ← AW When temp ÷ (mem16) > FFFFH (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % (mem16), AL ← temp ÷ (mem16)	1	1	1	1	0	1	1	1	mod		1	1	0		mem		35	2-4	u	u	u	u	u	u
		Signed Division Instructions																								
DIV	reg8	temp ← AW When temp ÷ reg8 > 0 and temp ÷ reg8 > 7FH or temp ÷ reg8 < 0 and temp ÷ reg8 < 0 - 7FH − 1 (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % reg8, AL ← temp ÷ reg8	1	1	1	1	0	1	1	0	1	1	1	1	1		reg		29-34	2	u	u	u	u	u	u

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	Flags AC	CY	V	P	S	Z
		Signed Division Instructions (cont)										
DIV	mem8	temp ← AW When temp ÷ (mem8) > 0 and (mem8) > 7FH or temp ÷ (mem8) < 0 and temp ÷ (mem8) < 0 - 7FH − 1 (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % (mem8), AL ← temp ÷ (mem8)	1 1 1 1 0 1 1 0	mod 1 1 1 mem	35-40	2-4	u	u	u	u	u	u
	reg16	temp ← AW When temp ÷ reg16 > 0 and reg16 > 7FFFH or temp ÷ reg16 < 0 and temp ÷ reg16 < 0 - 7FFFH − 1 (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % reg16, AL ← temp ÷ reg 16	1 1 1 1 0 1 1 1	1 1 1 1 reg	38-43	2	u	u	u	u	u	u
	mem16	temp ← AW When temp ÷ (mem16) > 0 and (mem16) > 7FFFH or temp ÷ (mem16) < 0 and temp ÷ (mem16) < 0 - 7FFFH − 1 (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % (mem16), AL ← temp ÷ (mem16)	1 1 1 1 0 1 1 1	mod 1 1 1 mem	48-53	2-4	u	u	u	u	u	u
		BCD Adjust Instructions										
ADJBA		When (AL AND 0FH) > 9 or AC = 1, AL ← AL + 6, AH ← AH + 1, AC ← 1, CY ← AC, AL ← AL AND 0FH	0 0 1 1 0 1 1 1		3	1	x	x	u	u	u	u
ADJ4A		When (AL AND 0FH) > 9 or AC = 1, AL ← AL + 6, CY ← CY OR AC, AC ← 1, When AL > 9FH, or CY = 1 AL ← AL + 60H, CY ← 1	0 0 1 0 0 1 1 1		3	1	x	x	u	x	x	x
ADJBS		When (AL AND 0FH) > 9 or AC = 1, AL ← AL − 6, AH ← AH − 1, AC ← 1, CY ← AC, AL ← AL AND 0FH	0 0 1 1 1 1 1 1		7	1	x	x	u	u	u	u
ADJ4S		When (AL AND 0FH) > 9 or AC = 1, AL ← AL − 6, CY ← CY OR AC, AC ← 1 When AL > 9FH or CY = 1 AL ← AL − 60H, CY ← 1	0 0 1 0 1 1 1 1		7	1	x	x	u	x	x	x

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	Flags AC	CY	V	P	S	Z
		Data Conversion Instructions										
CVTBD		AH ← AL ÷ 0AH, AL ← AL % 0AH	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0	15	2	u	u	u	x	x	x
CVTDB		AH ← 0, AL ← AH x 0AH + AL	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0	7	2	u	u	u	x	x	x
CVTBW		When AL < 80H, AH ← 0, all other times AH ← FFH	1 0 0 1 1 0 0 0		2	1						
CVTWL		When AL < 8000H, DW ← 0, all other times DW ← FFFFH	1 0 0 1 1 0 0 1		4-5	1						
		Comparison Instructions										
CMP	reg, reg	reg − reg	0 0 1 1 1 0 1 W	1 1 reg reg	2	2	x	x	x	x	x	x
	mem, reg	(mem) − reg	0 0 1 1 1 0 0 W	mod reg mem	11/15	2-4	x	x	x	x	x	x
	reg, mem	reg − (mem)	0 0 1 1 1 0 1 W	mod reg mem	11/15	2-4	x	x	x	x	x	x
	reg, imm	reg − imm	1 0 0 0 0 0 S W	1 1 1 1 1 reg	4	3-4	x	x	x	x	x	x
	mem, imm	(mem) − imm	1 0 0 0 0 0 S W	mod 1 1 1 mem	13/17	3-6	x	x	x	x	x	x
	acc, imm	When W = 0, AL − imm When W = 1, AW − imm	0 0 1 1 1 1 0 W		4	2-3	x	x	x	x	x	x
		Complement Instructions										
NOT	reg	reg ← $\overline{reg}$	1 1 1 1 0 1 1 W	1 1 0 1 0 reg	2	2						
	mem	(mem) ← ($\overline{mem}$)	1 1 1 1 0 1 1 W	mod 0 1 0 mem	16/24	2-4						
NEG	reg	reg ← $\overline{reg}$ + 1	1 1 1 1 0 1 1 W	1 1 0 1 1 reg	2	2	x	x	x	x	x	x
	mem	(mem) ← ($\overline{mem}$) + 1	1 1 1 1 0 1 1 W	mod 0 1 1 mem	16/24	2-4	x	x	x	x	x	x
		Logical Operation Instructions										
TEST	reg, reg	reg AND reg	1 0 0 0 0 1 0 W	1 1 reg reg	2	2	u	0	0	x	x	x
	mem, reg or reg, mem	(mem) AND reg	1 0 0 0 0 1 0 W	mod reg mem	10/14	2-4	u	0	0	x	x	x
	reg, imm	reg AND imm	1 1 1 1 0 1 1 W	1 1 0 0 0 reg	4	3-4	u	0	0	x	x	x
	mem, imm	(mem) AND imm	1 1 1 1 0 1 1 W	mod 0 0 0 mem	11/15	3-6	u	0	0	x	x	x
	acc, imm	When W = 0, AL AND imm8 When W = 1, AW AND imm8	1 0 1 0 1 0 0 W		4	2-3	u	0	0	x	x	x
AND	reg, reg	reg ← reg AND reg	0 0 1 0 0 0 1 W	1 1 reg reg	2	2	u	0	0	x	x	x
	mem, reg	(mem) ← (mem) AND reg	0 0 1 0 0 0 0 W	mod reg mem	16/24	2-4	u	0	0	x	x	x
	reg, mem	reg ← reg AND (mem)	0 0 1 0 0 0 1 W	mod reg mem	11/15	2-4	u	0	0	x	x	x
	reg, imm	reg ← reg AND imm	1 0 0 0 0 0 0 W	1 1 1 0 0 reg	4	3-4	u	0	0	x	x	x
	mem, imm	(mem) ← (mem) AND imm	1 0 0 0 0 0 0 W	mod 1 0 0 mem	18/26	3-6	u	0	0	x	x	x
	acc, imm	When W = 0, AL ← AL AND imm8 When W = 1, AW ← AW AND imm16	0 0 1 0 0 1 0 W		4	2-3	u	0	0	x	x	x

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	Flags AC	CY	V	P	S	Z
		Logical Operation Instructions (cont)										
OR	reg, reg	reg ← reg OR reg	0 0 0 0 1 0 1 W	1 1 reg reg	2	2	u	0	0	x	x	x
	mem, reg	(mem) ← (mem) OR reg	0 0 0 0 1 0 0 W	mod reg mem	16/24	2-4	u	0	0	x	x	x
	reg, mem	reg ← reg OR (mem)	0 0 0 0 1 0 1 W	mod reg mem	11/15	2-4	u	0	0	x	x	x
	reg, imm	reg ← reg OR imm	1 0 0 0 0 0 0 W	1 1 0 0 1 reg	4	3-4	u	0	0	x	x	x
	mem, imm	(mem) ← (mem) OR imm	1 0 0 0 0 0 0 W	mod 0 0 1 mem	18/26	3-6	u	0	0	x	x	x
	acc, imm	When W = 0, AL ← AL OR imm8 When W = 1, AW ← AW OR imm16	0 0 0 0 1 1 0 W		4	2-3	u	0	0	x	x	x
XOR	reg, reg	reg ← reg XOR reg	0 0 1 1 0 0 1 W	1 1 reg reg	2	2	u	0	0	x	x	x
	mem, reg	(mem) ← (mem) XOR reg	0 0 1 1 0 0 0 W	mod reg mem	16/24	2-4	u	0	0	x	x	x
	reg, mem	reg ← reg XOR (mem)	0 0 1 1 0 0 1 W	mod reg mem	11/15	2-4	u	0	0	x	x	x
	reg, imm	reg ← reg XOR imm	1 0 0 0 0 0 0 W	1 1 1 1 0 reg	4	3-4	u	0	0	x	x	x
	mem, imm	(mem) ← (mem) XOR imm	1 0 0 0 0 0 0 W	mod 1 1 0 mem	18/26	3-6	u	0	0	x	x	x
	acc, imm	When W = 0, AL ← AL XOR imm8 When W = 1, AW ← AW XOR imm16	0 0 1 1 0 1 0 W		4	2-3	u	0	0	x	x	x
		Bit Operation Instructions										
			2nd byte*	3rd byte*								
TEST1	reg8, CL	reg8 bit no. CL = 0: Z ← 1 reg8 bit no. CL = 1: Z ← 0	0 0 0 1 0 0 0 0	1 1 0 0 0 reg	3	3	u	0	0	u	u	x
	mem8, CL	(mem8) bit no. CL = 0: Z ← 1 (mem8) bit no. CL = 1: Z ← 0	0 0 0 1 0 0 0 0	mod 0 0 0 mem	12	3-5	u	0	0	u	u	x
	reg16, CL	reg16 bit no. CL = 0: Z ← 1 reg16 bit no. CL = 1: Z ← 0	0 0 0 1 0 0 0 1	1 1 0 0 0 reg	3	3	u	0	0	u	u	x
	mem16, CL	(mem16) bit no. CL = 0: Z ← 1 (mem16) bit no. CL = 1: Z ← 0	0 0 0 1 0 0 0 1	mod 0 0 0 mem	16	3-5	u	0	0	u	u	x
	reg8, imm3	reg8 bit no. imm3 = 0: Z ← 1 reg8 bit no. imm3 = 1: Z ← 0	0 0 0 1 1 0 0 0	1 1 0 0 0 reg	4	4	u	0	0	u	u	x
	mem8, imm3	(mem8) bit no. imm3 = 0: Z ← 1 (mem8) bit no. imm3 = 1: Z ← 0	0 0 0 1 1 0 0 0	mod 0 0 0 mem	13	4-6	u	0	0	u	u	x
	reg16, imm4	reg16 bit no. imm4 = 0: Z ← 1 reg16 bit no. imm4 = 1: Z ← 0	0 0 0 1 1 0 0 1	1 1 0 0 0 reg	4	4	u	0	0	u	u	x
	mem16, imm4	(mem16) bit no. imm4 = 0: Z ← 1 (mem16) bit no. imm4 = 1: Z ← 0	0 0 0 1 1 0 0 1	mod 0 0 0 mem	17	4-6	u	0	0	u	u	x
			2nd byte*	3rd byte*								

*Note: First byte = 0FH

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	Flags AC	CY	V	P	S	Z
		Bit Operation Instructions (cont)										
			2nd byte*	3rd byte*								
NOT1	reg8, CL	reg8 bit no. CL ← $\overline{\text{reg8 bit no. CL}}$	0 0 0 1 0 1 1 0	1 1 0 0 0 reg	4	3						
	mem8, CL	(mem8) bit no. CL ← $\overline{\text{(mem8) bit no. CL}}$	0 0 0 1 0 1 1 0	mod 0 0 0 mem	18	3-5						
	reg16, CL	reg16 bit no. CL ← $\overline{\text{reg16 bit no. CL}}$	0 0 0 1 0 1 1 1	1 1 0 0 0 reg	4	3						
	mem16, CL	(mem16) bit no. CL ← $\overline{\text{(mem16) bit no. CL}}$	0 0 0 1 0 1 1 1	mod 0 0 0 mem	26	3-5						
	reg8, imm3	reg8 bit no. imm3 ← $\overline{\text{reg8 bit no. imm3}}$	0 0 0 1 1 1 1 0	1 1 0 0 0 reg	5	4						
	mem8, imm3	(mem8) bit no. imm3 ← $\overline{\text{(mem8) bit no. imm3}}$	0 0 0 1 1 1 1 0	mod 0 0 0 mem	19	4-6						
	reg16, imm4	reg16 bit no. imm4 ← $\overline{\text{(reg16) bit no. imm4}}$	0 0 0 1 1 1 1 1	1 1 0 0 0 reg	5	4						
	mem16, imm4	(mem16) bit no. imm4 ← $\overline{\text{(mem16) bit no. imm4}}$	0 0 0 1 1 1 1 1	mod 0 0 0 mem	27	4-6						
			2nd byte*	3rd byte*								
			*Note: First byte = 0FH									
	CY	CY ← $\overline{\text{CY}}$	1 1 1 1 0 1 0 1		2	1		x				
			2nd byte*	3rd byte*								
CLR1	reg8, CL	reg8 bit no. CL ← 0	0 0 0 1 0 0 1 0	1 1 0 0 0 reg	5	3						
	mem8, CL	(mem8) bit no. CL ← 0	0 0 0 1 0 0 1 0	mod 0 0 0 mem	14	3-5						
	reg16, CL	reg16 bit no. CL ← 0	0 0 0 1 0 0 1 1	1 1 0 0 0 reg	5	3						
	mem16, CL	(mem16) bit no. CL ← 0	0 0 0 1 0 0 1 1	mod 0 0 0 mem	22	3-5						
	reg8, imm3	reg8 bit no. imm3 ← 0	0 0 0 1 1 0 1 0	1 1 0 0 0 reg	6	4						
	mem8, imm3	(mem8) bit no. imm3 ← 0	0 0 0 1 1 0 1 0	mod 0 0 0 mem	15	4-6						
	reg16, imm4	reg16 bit no. imm4 ← 0	0 0 0 1 1 0 1 1	1 1 0 0 0 reg	6	4						
	mem16, imm4	(mem16) bit no. imm4 ← 0	0 0 0 1 1 0 1 1	mod 0 0 0 mem	27	4-6						
			2nd byte*	3rd byte*								
			*Note: First byte = 0FH									
	CY	CY ← 0	1 1 1 1 1 0 0 0		2	1		0				
	DIR	DIR ← 0	1 1 1 1 1 1 0 0		2	1						

NEC

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	Flags AC	CY	V	P	S	Z
		Bit Operation Instructions (cont)										
SET1	reg8, CL	reg8 bit no. CL ← 1	0 0 0 1 0 1 0 0	1 1 0 0 0 reg	4	3						
	mem8, CL	(mem8) bit no. CL ← 1	0 0 0 1 0 1 0 0	mod 0 0 0 mem	13	3-5						
	reg16, CL	reg16 bit no. CL ← 1	0 0 0 1 0 1 0 1	1 1 0 0 0 reg	4	3						
	mem16, CL	(mem16) bit no. CL ← 1	0 0 0 1 0 1 0 1	mod 0 0 0 mem	21	3-5						
	reg8, imm3	reg8 bit no. imm3 ← 1	0 0 0 1 1 1 0 0	1 1 0 0 0 reg	5	4						
	mem8, imm3	(mem8) bit no. imm3 ← 1	0 0 0 1 1 1 0 0	mod 0 0 0 mem	14	4-6						
	reg16, imm4	reg16 bit no. imm4 ← 1	0 0 0 1 1 1 0 1	1 1 0 0 0 reg	5	4						
	mem16, imm4	(mem16) bit no. imm4 ← 1	0 0 0 1 1 1 0 1	mod 0 0 0 mem	22	4-6						
			2nd byte*	3rd byte*								
			*Note: First byte = 0FH									
	CY	CY ← 1	1 1 1 1 1 0 0 1		2	1		1				
	DIR	DIR ← 1	1 1 1 1 1 1 0 1		2	1						
		Shift Instructions										
SHL	reg, 1	CY ← MSB of reg, reg ← reg x 2 When MSB of reg ≠ CY, V ← 1 When MSB of reg = CY, V ← 0	1 1 0 1 0 0 0 W	1 1 1 0 0 reg	2	2	u	x	x	x	x	x
	mem, 1	CY ← MSB of (mem), (mem) ← (mem) x 2 When MSB of (mem) ≠ CY, V ← 1 When MSB of (mem) = CY, V ← 0	1 1 0 1 0 0 0 W	mod 1 0 0 mem	16/24	2-4	u	x	x	x	x	x
	reg, CL	temp ← CL, while temp ≠ 0; repeat this operation, CY ← MSB of reg, reg ← reg x 2, temp ← temp − 1	1 1 0 1 0 0 1 W	1 1 1 0 0 reg	7 + n	2	u	x	u	x	x	x
	mem, CL	temp ← CL, while temp ≠ 0, repeat this operation, CY ← MSB of (mem), (mem) ← (mem) x 2, temp ← temp − 1	1 1 0 1 0 0 1 W	mod 1 0 0 mem	19/27 + n	2-4	u	x	u	x	x	x
	reg, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, CY ← MSB of reg, reg ← reg x 2, temp ← temp − 1	1 1 0 0 0 0 0 W	1 1 1 0 0 reg	7 + n	3	u	x	u	x	x	x
	mem, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, CY ← MSB of (mem), (mem) ← (mem) x 2, temp ← temp − 1	1 1 0 0 0 0 0 W	mod 1 0 0 mem	19/27 + n	3-5	u	x	u	x	x	x
			n: number of shifts									
SHR	reg, 1	CY ← LSB of reg, reg ← reg ÷ 2 When MSB of reg ≠ bit following MSB of reg: V ← 1 When MSB of reg = bit following MSB of reg: V ← 0	1 1 0 1 0 0 0 W	1 1 1 0 1 reg	2	2	u	x	x	x	x	x

Mnemonic	Operand	Operation	Operation Code 7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	No. of Clocks	No. of Bytes	Flags AC	CY	V	P	S	Z
		Shift Instructions (cont)																								
SHR	mem, 1	CY ← LSB of (mem), (mem) ← (mem) ÷ 2 When MSB of (mem) ≠ bit following MSB of (mem): V ← 1 When MSB of (mem) = bit following MSB of (mem): V ← 0	1	1	0	1	0	0	0	W	mod		1	0	1	mem			16/24	2-4	u	x	x	x	x	x
	reg, CL	temp ← CL, while temp ≠ 0, repeat this operation, CY ← LSB of reg, reg ← reg ÷ 2, temp ← temp − 1	1	1	0	1	0	0	0	W	1	1	1	0	1	reg			7 + n	2	u	x	u	x	x	x
	mem, CL	temp ← CL, while temp ≠ 0, repeat this operation, CY ← LSB of (mem), (mem) ← (mem) ÷ 2, temp ← temp − 1	1	1	0	1	0	0	1	W	mod		1	0	1	mem			19/27+n	2-4	u	x	u	x	x	x
	reg, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, CY ← LSB of reg, reg ← reg ÷ 2, temp ← temp − 1	1	1	0	0	0	0	0	W	1	1	1	0	1	reg			7 + n	3	u	x	u	x	x	x
	mem, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, CY ← LSB of (mem), (mem) ← (mem) ÷ 2, temp ← temp − 1 n: number of shifts	1	1	0	0	0	0	0	W	mod		1	0	1	mem			19/27+n	3-5	u	x	u	x	x	x
SHRA	reg, 1	CY ← LSB of reg, reg ← reg ÷ 2, V ← 0 MSB of operand does not change	1	1	0	1	0	0	0	W	1	1	1	1	1	reg			2	2	u	x	0	x	x	x
	mem, 1	CY ← LSB of (mem), (mem) ← (mem) ÷ 2, V ← 0, MSB of operand does not change	1	1	0	1	0	0	0	W	mod		1	1	1	mem			16/24	2-4	u	x	0	x	x	x
	reg, CL	temp ← CL, while temp ≠ 0, repeat this operation, CY ← LSB of reg, reg ← reg ÷ 2, temp ← temp − 1 MSB of operand does not change	1	1	0	1	0	0	1	W	1	1	1	1	1	reg			7 + n	2	u	x	u	x	x	x
	mem, CL	temp ← CL, while temp ≠ 0, repeat this operation, CY ← LSB of (mem), (mem) ← (mem) ÷ 2, temp ← temp − 1 MSB of operand does not change	1	1	0	1	0	0	1	W	mod		1	1	1	mem			19/27+n	2-4	u	x	u	x	x	x
	reg, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, CY ← LSB of reg, reg ← reg ÷ 2, temp ← temp − 1 MSB of operand does not change	1	1	0	0	0	0	0	W	1	1	1	1	1	reg			7 + n	3	u	x	u	x	x	x
	mem, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, CY ← LSB of (mem), (mem) ← (mem) ÷ 2, temp ← temp − 1 MSB of operand does not change n: number of shifts	1	1	0	0	0	0	0	W	mod		1	1	1	mem			19/27+n	3-5	u	x	u	x	x	x

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	Flags AC	CY	V	P	S	Z
		Rotation Instructions										
ROL	reg, 1	CY ← MSB of reg, reg ← reg x 2 + CY MSB of reg ≠ CY: V ← 1 MSB of reg = CY: V ← 0	1 1 0 1 0 0 0 W	1 1 0 0 0 reg	2	2		x	x			
	mem, 1	CY ← MSB of (mem), (mem) ← (mem) x 2 + CY MSB of (mem) ≠ CY: V ← 1 MSB of (mem) = CY: V ← 0	1 1 0 1 0 0 0 W	mod 0 0 0 mem	16/24	2-4		x	x			
	reg, CL	temp ← CL, while temp ≠ 0, repeat this operation, CY ← MSB of reg, reg ← reg x 2 + CY temp ← temp − 1	1 1 0 1 0 0 1 W	1 1 0 0 0 reg	7 + n	2		x	u			
	mem, CL	temp ← CL, while temp ≠ 0, repeat this operation, CY ← MSB of (mem), (mem) ← (mem) x 2 + CY temp ← temp − 1	1 1 0 1 0 0 1 W	mod 0 0 0 reg	19/27+n	2-4		x	u			
	reg, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, CY ← MSB of reg, reg ← reg x 2 + CY temp ← temp − 1	1 1 0 0 0 0 0 W	1 1 0 0 0 reg	7 + n	3		x	u			
	mem, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, CY ← MSB of (mem), (mem) ← (mem) x 2 + CY temp ← temp − 1	1 1 0 0 0 0 0 W	mod 0 0 0 mem	19/27+n	3-5		x	u			
			n: number of shifts									
ROR	reg, 1	CY ← LSB of reg, reg ← reg ÷ 2 MSB of reg ← CY MSB of reg ≠ bit following MSB of reg: V ← 1 MSB of reg = bit following MSB of reg: V ← 0	1 1 0 1 0 0 0 W	1 1 0 0 1 reg	2	2		x	x			
	mem, 1	CY ← LSB of (mem), (mem) ← (mem) ÷ 2 MSB of (mem) ← CY MSB of (mem) ≠ bit following MSB of (mem): V ← 1 MSB of (mem) = bit following MSB of (mem): V ← 0	1 1 0 1 0 0 0 W	mod 0 0 1 mem	16/24	2-4		x	x			
	reg, CL	temp ← CL, while temp ≠ 0, repeat this operation, CY ← LSB of reg, reg ← reg ÷ 2, MSB of reg ← CY temp ← temp − 1	1 1 0 1 0 0 1 W	1 1 0 0 1 reg	7 + n	2		x	u			
	mem, CL	temp ← CL, while temp ≠ 0, repeat this operation, CY ← LSB of (mem), (mem) ← (mem) ÷ 2, MSB of (mem) ← CY temp ← temp − 1	1 1 0 1 0 0 1 W	mod 0 0 1 mem	19/27+n	2-4		x	u			
			n:number of shifts									

Mnemonic	Operand	Operation	Operation Code 7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	No. of Clocks	No. of Bytes	Flags AC	CY	V	P	S	Z
		Rotation Instructions (cont)																								
ROR	reg, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, CY ← LSB of reg, reg ← reg ÷ 2, MSB of reg ← CY temp ← temp − 1	1	1	0	0	0	0	0	W	1	1	0	0	1		reg		7 + n	3		x	u			
	mem, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, CY ← LSB of (mem), (mem) ← (mem) ÷ 2 temp ← temp − 1	1	1	0	0	0	0	0	W	mod		0	0	1		mem		19/27+n	3-5		x	u			
		n: number of shifts																								
		Rotate Instructions																								
ROLC	reg, 1	tmpcy ← CY, CY ← MSB of reg reg ← reg x 2 + tmpcy MSB of reg = CY: V ← 0 MSB of reg ≠ CY: V ← 1	1	1	0	1	0	0	0	W	1	1	0	1	0		reg		2	2		x	x			
	mem, 1	tmpcy ← CY, CY ← MSB of (mem) (mem) ← (mem) x 2 + tmpcy MSB of (mem) = CY: V ← 0 MSB of (mem) ≠ CY: V ← 1	1	1	0	1	0	0	0	W	mod		0	1	0		mem		16/24	2-4		x	x			
	reg, CL	temp ← CL, while temp ≠ 0, repeat this operation, tmpcy ← CY, CY ← MSB of reg, reg ← reg x 2 + tmpcy temp ← temp − 1	1	1	0	1	0	0	1	W	1	1	0	1	0		reg		7 + n	2		x	u			
	mem, CL	temp ← CL, while temp ≠ 0, repeat this operation, tmpcy ← CY, CY ← MSB of (mem), (mem) ← (mem) x 2 + tmpcy temp ← temp − 1	1	1	0	1	0	0	1	W	mod		0	1	0		mem		19/27+n	2-4		x	u			
	reg, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, tmpcy ← CY, CY ← MSB of reg, reg ← reg x 2 + tmpcy temp ← temp − 1	1	1	0	0	0	0	0	W	1	1	0	1	0		reg		7 + n	3		x	u			
	mem, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, tmpcy ← CY, CY ← MSB of (mem) (mem) ← (mem) x 2 + tmpcy temp ← temp − 1	1	1	0	0	0	0	0	W	mod		0	1	0		mem		19/27+n	3-5		x	u			
		n: number of shifts																								

Mnemonic	Operand	Operation	Operation Code 7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	No. of Clocks	No. of Bytes	Flags AC	CY	V	P	S	Z
		Rotate Instructions (cont)																								
RORC	reg, 1	tmpcy ← CY, CY ← LSB of reg reg ← reg ÷ 2, MSB of reg ← tmpcy MSB of reg ≠ bit following MSB of reg: V ← 1 MSB of reg = bit following MSB of reg: V ← 0	1	1	0	1	0	0	0	W	1	1	1	0	1	reg			2	2		x	x			
	mem, 1	tmpcy ← CY, CY ← LSB of (mem) (mem) ← (mem) ÷ 2, MSB of (mem) ← tmpcy MSB of (mem) ≠ bit following MSB of (mem): V ← 1 MSB of (mem) = bit following MSB of (mem): V ← 0	1	1	0	1	0	0	0	W	mod		0	1	1	mem			16/24	2-4		x	x			
	reg, CL	temp ← CL, while temp ≠ 0, repeat this operation, tmpcy ← CY, CY ← LSB of reg, reg ← reg ÷ 2, MSB of reg ← tmpcy, temp ← temp − 1	1	1	0	1	0	0	1	W	1	1	0	1	1	reg			7 + n	2		x	u			
	mem, CL	temp ← CL, while temp ≠ 0, repeat this operation, tmpcy ← CY, CY ← LSB of (mem), (mem) ← (mem) ÷ 2 MSB of (mem) ← tmpcy, temp ← temp − 1	1	1	0	1	0	0	1	W	mod		0	1	1	mem			19/27 + n	2-4		x	u			
	reg, imm8	temp ← imm8, while temp ≠ 0 repeat this operation, tmpcy ← CY, CY ← LSB of reg, reg ← reg ÷ 2 MSB of reg ← tmpcy, temp ← temp − 1	1	1	0	0	0	0	0	W	1	1	0	1	1	reg			7 + n	3		x	u			
	mem, imm8	temp ← imm8, while temp ≠ 0, repeat this operation, tmpcy ← CY, CY ← LSB of (mem), (mem) ← (mem) ÷ 2 MSB of (mem) ← tmpcy, temp ← temp − 1	1	1	0	0	0	0	0	W	mod		0	1	1	mem			19/27 + n	3-5		x	u			
																			n: number of shifts							
		Subroutine Control Instructions																								
CALL	near-proc	(SP − 1, SP − 2) ← PC, SP ← SP − 2 PC ← PC + disp	1	1	1	0	1	0	0	0									20	3						
	regptr16	(SP − 1, SP − 2) ← PC, SP ← SP − 2 PC ← regptr16	1	1	1	1	1	1	1	1	1	1	0	1	0	reg			18	2						
	memptr16	(SP − 1, SP − 2) ← PC, SP ← SP − 2 PC ← (memptr16)	1	1	1	1	1	1	1	1	mod		0	1	0	mem			31	2-4						
	far-proc	(SP − 1, SP − 2) ← PS, (SP − 3, SP − 4) ← PC SP ← SP − 4, PS ← seg, PC ← offset	1	0	0	1	1	0	1	0									29	5						
	memptr32	(SP − 1, SP − 2) ← PS, (SP − 3, SP − 4) ← PC SP ← SP − 4, PS ← (memptr32 + 2), PC ← (memptr32)	1	1	1	1	1	1	1	1	mod		0	1	1	mem			47	2-4						

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	Flags AC	CY	V	P	S	Z
		Subroutine Control Instructions (cont)										
RET		PC ← (SP + 1, SP), SP ← SP + 2	1 1 0 0 0 0 1 1		19	1						
	pop-value	PC ← (SP + 1, SP) SP ← SP + 2, SP ← SP + pop-value	1 1 0 0 0 0 1 0		24	3						
		PC ← (SP + 1, SP), PS ← (SP + 3, SP + 2) SP ← SP + 4	1 1 0 0 1 0 1 1		29	1						
	pop-value	PC ← (SP + 1, SP), PS ← (SP + 3, SP + 2) SP ← SP + 4, SP ← SP + pop-value	1 1 0 0 1 0 1 0		32	3						
		Stack Manipulation Instructions										
PUSH	mem16	(SP − 1, SP − 2) ← (mem16), SP ← SP − 2	1 1 1 1 1 1 1 1	mod 1 1 0 mem	26	2-4						
	reg16	(SP − 1, SP − 2) ← reg16, SP ← SP − 2	0 1 0 1 0 reg		12	1						
	sreg	(SP− 1, SP− 2) ← sreg, SP ← SP − 2	0 0 0 sreg 1 1 0		12	1						
	PSW	(SP − 1, SP − 2) ← PSW, SP ← SP − 2	1 0 0 1 1 1 0 0		12	1						
	R	Push registers on the stack	0 1 1 0 0 0 0 0		67	1						
	imm	(SP − 1, SP − 2) ← imm SP ← SP − 2, When S = 1, sign extension	0 1 1 0 1 0 S 0		11/ 12	2-3						
POP	mem16	(mem16) ← (SP + 1, SP), SP ← SP + 2	1 0 0 0 1 1 1 1	mod 0 0 0 mem	25	2-4						
	reg16	reg16 ← (SP + 1, SP), SP ← SP + 2	0 1 0 1 1 reg		12	1						
	sreg	sreg ← (SP + 1, SP) sreg : SS, DS0, DS1 SP ← SP + 2	0 0 0 sreg 1 1 1		12	1						
	PSW	PSW ← (SP + 1, SP), SP ← SP + 2	1 0 0 1 1 1 0 1		12	1	R	R	R	R	R	R
	R	Pop registers from the stack	0 1 1 0 0 0 0 1		75	1						
PREPARE	imm16, imm8	Prepare new stack frame	1 1 0 0 1 0 0 0 *: imm8 = 0: 16 imm8 > 1: 23 + 16 (imm8 − 1)		*	4						
DISPOSE		Dispose of stack frame	1 1 0 0 1 0 0 1		10	1						
		Branch Instruction										
BR	near-label	PC ← PC + disp	1 1 1 0 1 0 0 1		13	3						
	short-label	PC ← PC + ext-disp8	1 1 1 0 1 0 1 1		12	2						
	regptr16	PC ← regptr16	1 1 1 1 1 1 1 1	1 1 1 0 0 reg	11	2						
	memptr16	PC ← (memptr16)	1 1 1 1 1 1 1 1	mod 1 0 0 mem	24	2-4						
	far-label	PS ← seg, PC ← offset	1 1 1 0 1 0 1 0		15	5						
	memptr32	PS ← (memptr32 + 2), PC ← (memptr32)	1 1 1 1 1 1 1 1	mod 1 0 1 mem	35	2-4						

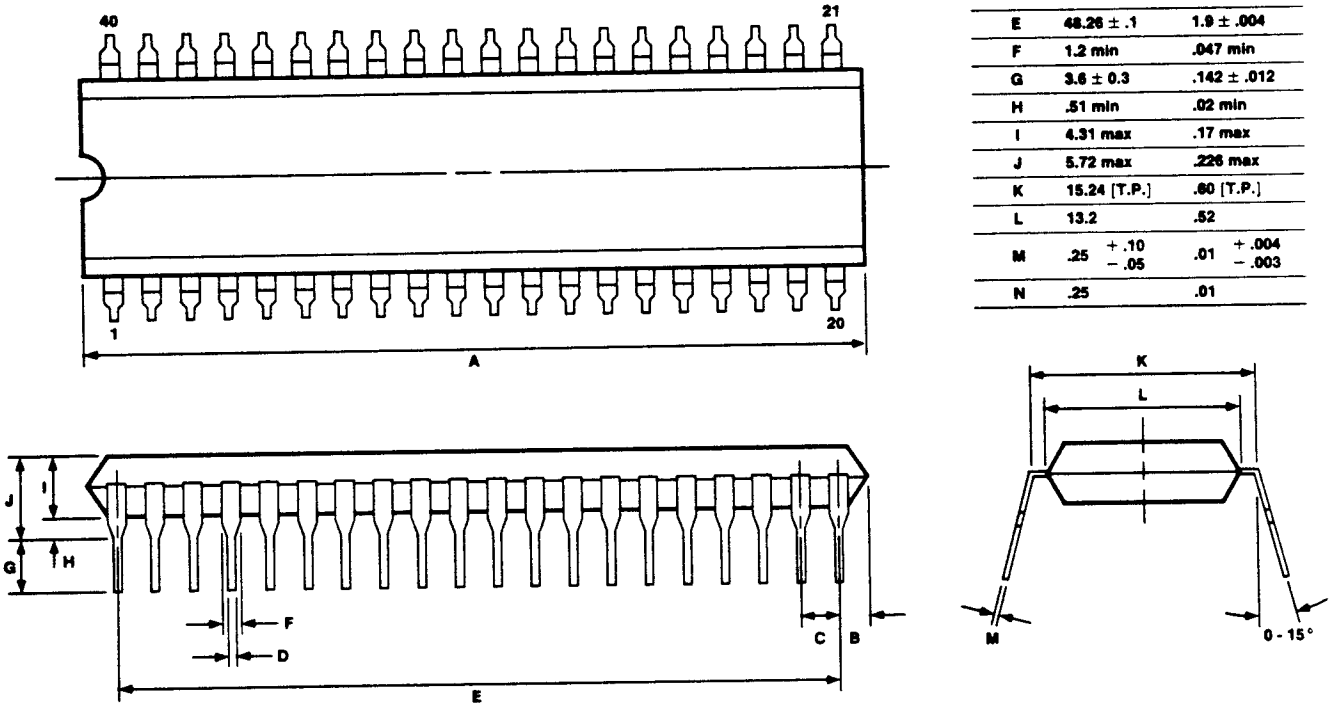
NEC

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	Flags AC	CY	V	P	S	Z
		Conditional Branch Instructions									
BV	short-label	if V = 1, PC ← PC + ext-disp8	0 1 1 1 0 0 0 0	14/4	2						
BNV	short-label	if V = 0, PC ← PC + ext-disp8	0 1 1 1 0 0 0 1	14/4	2						
BC, BL	short-label	if CY = 1, PC ← PC + ext-disp8	0 1 1 1 0 0 1 0	14/4	2						
BNC, BNL	short-label	if CY = 0, PC ← PC + ext-disp8	0 1 1 1 0 0 1 1	14/4	2						
BE, BZ	short-label	if Z = 1, PC ← PC + ext-disp8	0 1 1 1 0 1 0 0	14/4	2						
BNE, BNZ	short-label	if Z = 0, PC ← PC + ext-disp8	0 1 1 1 0 1 0 1	14/4	2						
BNH	short-label	if CY OR Z = 1, PC ← PC + ext-disp8	0 1 1 1 0 1 1 0	14/4	2						
BH	short-label	if CY OR Z = 0, PC ← PC + ext-disp8	0 1 1 1 0 1 1 1	14/4	2						
BN	short-label	if S = 1, PC ← PC + ext-disp8	0 1 1 1 1 0 0 0	14/4	2						
BP	short-label	if S = 0, PC ← PC + ext-disp8	0 1 1 1 1 0 0 1	14/4	2						
BPE	short-label	if P = 1, PC ← PC + ext-disp8	0 1 1 1 1 0 1 0	14/4	2						
BPO	short-label	if P = 0, PC ← PC + ext-disp8	0 1 1 1 1 0 1 1	14/4	2						
BLT	short-label	if S XOR V = 1, PC ← PC + ext-disp8	0 1 1 1 1 1 0 0	14/4	2						
BGE	short-label	if S XOR V = 0, PC ← PC + ext-disp8	0 1 1 1 1 1 0 1	14/4	2						
BLE	short-label	if (S XOR V) OR Z = 1, PC ← PC + ext-disp8	0 1 1 1 1 1 1 0	14/4	2						
BGT	short-label	if (S XOR V) OR Z = 0, PC ← PC + ext-disp8	0 1 1 1 1 1 1 1	14/4	2						
DBNZNE	short-label	CW ← CW − 1 if Z = 0 and CW ≠ 0, PC ← PC + ext-disp8	1 1 1 0 0 0 0 0	14/5	2						
DBNZE	short-label	CW ← CW − 1 if Z = 1 and CW ≠ 0, PC ← PC + ext-disp8	1 1 1 0 0 0 0 1	14/5	2						
DBNZ	short-label	CW ← CW − 1 if CW ≠ 0, PC ← PC + ext-disp8	1 1 1 0 0 0 1 0	13/5	2						
BCWZ	short-label	if CW = 0, PC ← PC + ext-disp8	1 1 1 0 0 0 1 1	13/5	2						
		Interrupt Instructions									
BRK	3	(SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS, (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0 PS ← (15, 14), PC ← (13, 12)	1 1 0 0 1 1 0 0	50	1						
	imm8 (≠ 3)	(SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS, (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0 PC ← (n x 4, + 1, n x 4) PS ← (n x 4 + 3, n x 4 + 2) n = imm8	1 1 0 0 1 1 0 1	50	2						

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	Flags AC	CY	V	P	S	Z
		Interrupt Instructions (cont)										
BRKV		When V = 1 (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS, (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0 PS ← (19, 18), PC ← (17, 16)	1 1 0 0 1 1 1 0		52/3	1						
RETI		PC ← (SP + 1, SP), PS ← (SP + 3, SP + 2), PSW ← (SP + 5, SP + 4), SP ← SP + 6	1 1 0 0 1 1 1 1		39	1	R	R	R	R	R	R
CHKIND	reg16, mem32	When (mem32) > reg16 or (mem32 + 2) < reg16 (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS, (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0, PS ← (23, 22), PC ← (21, 20)	0 1 1 0 0 0 1 0	mod reg mem	73-76/ 26	2-4						
BRKEM	imm8	(SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS, (SP − 5, SP − 6) ← PC, SP ← SP − 6 MD ← 0, PC ← (n x 4 + 1, n x 4), MD Bit Write Enable PS ← (n x 4 + 3, n x 4 + 2), n = imm8	0 0 0 0 1 1 1 1	1 1 1 1 1 1 1 1	50	3						
		CPU Control Instructions										
HALT		CPU Halt	1 1 1 1 0 1 0 0		2	1						
BUSLOCK		Bus Lock Prefix	1 1 1 1 0 0 0 0		2	1						
FPO1	fp-op	No Operation	1 1 0 1 1 X X X	1 1 Y Y Y Z Z Z	2	2						
	fp-op, mem	data bus ← (mem)	1 1 0 1 1 X X X	mod Y Y Y mem	15	2-4						
FPO2	fp-op	No Operation	0 1 1 0 0 1 1 X	1 1 Y Y Y Z Z Z	2	2						
	fp-op, mem	data bus ← (mem)	0 1 1 0 0 1 1 X	mod Y Y Y mem	15	2-4						
POLL		Poll and wait	1 0 0 1 1 0 1 1	n: number of times POLL pin is sampled	2 + 5n	1						
NOP		No Operation	1 0 0 1 0 0 0 0		3	1						
DI		IE ← 0	1 1 1 1 1 0 1 0		2	1						
EI		IE ← 1	1 1 1 1 1 0 1 1		2	1						
		8080 Mode Instructions										
RETEM		PC ← (SP + 1, SP), PS ← (SP + 3, SP + 2), PSW ← (SP + 5, SP + 4), SP ← SP + 6, MD Bit Write Disable	1 1 1 0 1 1 0 1	1 1 1 1 1 1 0 1	39	2	R	R	R	R	R	R
CALLN	imm8	(SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS, (SP − 5, SP − 6) ← PC, SP ← SP − 6 MD ← 1, PC ← (n x 4 + 1, n x 4) PS ← (n x 4 + 3, n x 4 + 2), n = imm8	1 1 1 0 1 1 0 1	1 1 1 1 0 1 1 0 1	58	3						

Packaging Information

40-Pin Plastic DIP Package (600 mil)



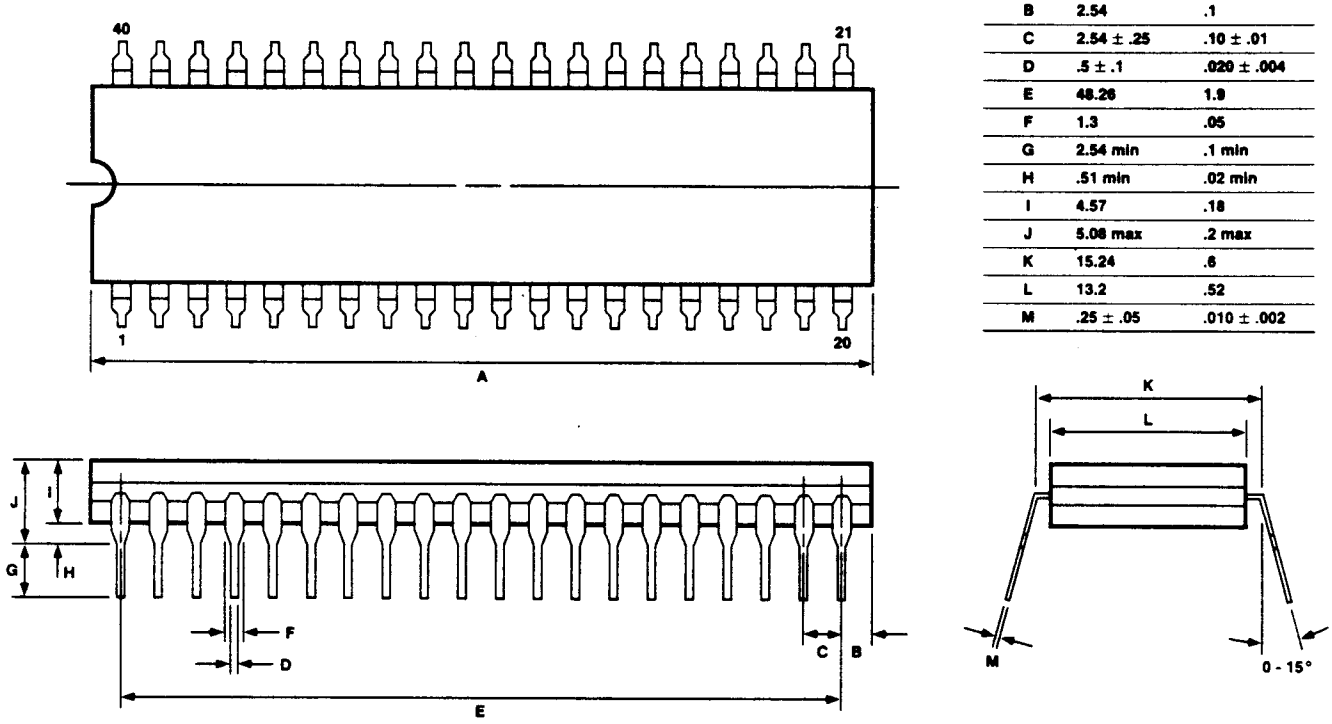
Item	Millimeters	Inches
A	53.34 max	2.1 max
B	2.54 max	.10 max
C	2.54 [T.P.]	.10 [T.P.]
D	.5 ± .10	.02 $^{+.004}_{-.005}$
E	48.26 ± .1	1.9 ± .004
F	1.2 min	.047 min
G	3.6 ± 0.3	.142 ± .012
H	.51 min	.02 min
I	4.31 max	.17 max
J	5.72 max	.226 max
K	15.24 [T.P.]	.60 [T.P.]
L	13.2	.52
M	.25 $^{+.10}_{-.05}$	.01 $^{+.004}_{-.003}$
N	.25	.01

Notes: 1. Each lead centerline is located within 0.25 mm [0.01 inch] of its true position [T.P.] at maximum material condition.
2. Item "K" to center of leads when formed parallel.

83-001399B

Packaging Information (cont)

40-Pin Cerdip Package



Item	Millimeters	Inches
A	53.34 max	2.1 max
B	2.54	.1
C	2.54 ± .25	.10 ± .01
D	.5 ± .1	.020 ± .004
E	48.26	1.9
F	1.3	.05
G	2.54 min	.1 min
H	.51 min	.02 min
I	4.57	.18
J	5.08 max	.2 max
K	15.24	.6
L	13.2	.52
M	.25 ± .05	.010 ± .002

83-001400B

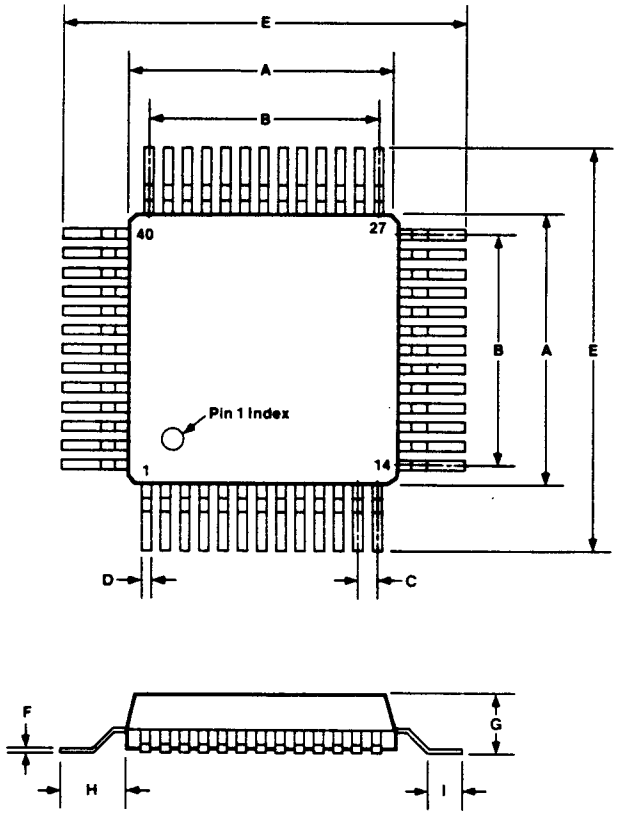
Packaging Information (cont)

44-Pin Plastic Leadless Chip Carrier (PLCC) Package

(Information available in first quarter of 1986.)

Packaging Information (cont)

52-Pin Plastic Miniflat Package



Item	Millimeters	Inches
A	$14.0^{+.3}_{-.2}$	$.551^{+.012}_{-.008}$
B	12.0 ±.3	.472 ±.012
C	1.00 ±.15	.039 ±.006
D	$.4^{+.2}_{-.1}$	$.016^{+.008}_{-.004}$
E	21.0 ±.4	.827 ±.016
F	$.15^{+.10}_{-.05}$	$.006^{+.004}_{-.002}$
G	2.8 max	.110 max
H	3.3 ±.2	.130 ±.008
I	2.2 ±.2	.087 ±.008

83-001875B

Notes:

REGIONAL SALES AND ENGINEERING SUPPORT OFFICES

NORTHEAST
Twenty Burlington Mall Road, Suite 449
Burlington, MA 01803
TEL 617-272-1774 TWX 710-348-6515

SOUTHEAST
Radice Corporate Center
600 Corporate Drive, Suite 412
Fort Lauderdale, FL 33334
TEL 305-776-0682 TWX 759839

MIDWEST
3025 West Salt Creek Lane, Suite 300
Arlington Heights, IL 60005
TEL 312-577-9090 TWX 910-687-1492

SOUTHCENTRAL
16475 Dallas Parkway, Suite 380
Dallas, TX 75248
TEL 214-931-0641 TWX 910-860-5284

SOUTHWEST
200 East Sandpointe, Building 8 Suite 460
Santa Ana, CA 92707
TEL 714-546-0501 TWX 759845

NORTHWEST
10080 North Wolfe Road, SW3 Suite 360
Cupertino, CA 95014
TEL 408-446-0650 TLX 595497

DISTRICT OFFICES

200 Broadhollow Road, Suite 302
Route 110
Melville, NY 11747
TEL 516-423-2500 TWX 510-224-6090

Beechwood Office Park
385 South Road
Poughkeepsie, NY 12601
TEL 914-452-4747 TWX 510-248-0066

200 Perinton Hills Office Park
Fairport, NY 14450
TEL 716-425-4590 TWX 510-100-8949

5720 Peachtree Parkway, Suite 120
Norcross, GA 30092
TEL 404-447-4409 TWX 910-997-0450

7257 Parkway Drive, Suite 109
Hanover, MD 21076
TEL 301-796-3944 TLX 759847

29200 Southfield Road, Suite 208
Southfield, MI 48076
TEL 313-559-4242 TWX 810-224-4625

Busch Corporate Center
6480 Busch Blvd., Suite 121
Columbus, OH 43229
TEL 614-436-1778 TWX 510-101-1771

8030 Cedar Avenue South, Suite 229
Bloomington, MN 55420
TEL 612-854-4443 TWX 910-997-0726

DISTRICT OFFICES [cont]

Echelon Building 2
9430 Research Boulevard, Suite 330
Austin, TX 78759
TEL 512-346-9280

6150 Canoga Avenue, Suite 112
Woodland Hills, CA 91367
TEL 818-716-1535 TWX 559210

Lincoln Center Building
10300 S.W. Greenburg Road, Suite 540
Portland, OR 97223
TEL 503-245-1600

5445 DTC Parkway, Suite 218
Englewood, CO 80111
TEL 303-694-0041 TWX 510-600-5666

NATICK TECHNOLOGY CENTER

One Natick Executive Park
Natick, MA 01760
TEL 617-655-8833 TWX 710-386-2110

NEC
NEC Electronics Inc.
CORPORATE HEADQUARTERS
401 Ellis Street
P.O. Box 7241
Mountain View, CA 94039
TEL 415-960-6000
TWX 910-379-6985

NECEL-000010-1185
STOCK NO. 500635

OKI MSM82C51A UART Data Sheet

OKI semiconductor

JUNE 1984

MSM82C51A
UNIVERSAL SYNCHRONOUS ASYNCHRONOUS RECEIVER TRANSMITTER

GENERAL DESCRIPTION

The MSM82C51A is a USART (Universal Synchronous Asynchronous Receiver Transmitter) for serial data communication for the microcomputer systems.

The MSM82C51A receives parallel data from the CPU and transmits serial data. This device also receives serial data and transmits parallel data to the CPU.

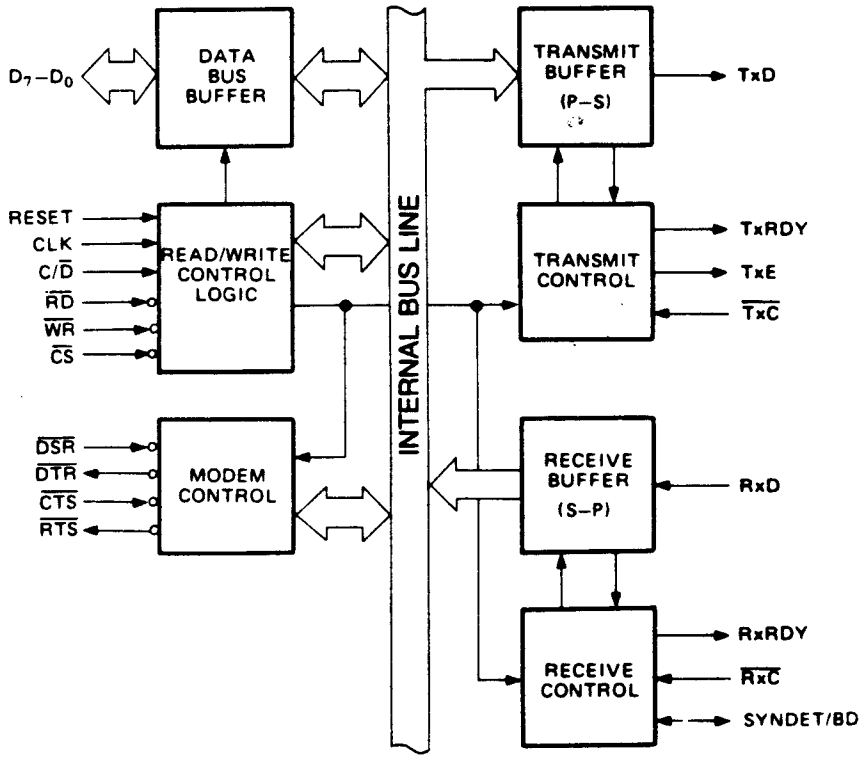
The MSM82C51A is a fully static circuit using silicon gate CMOS technology. It operates on an extremely low power supply at 100 μA (max) of standby current by suspending all the operations.

MSM82C51A is functionally compatible with the 8251A.

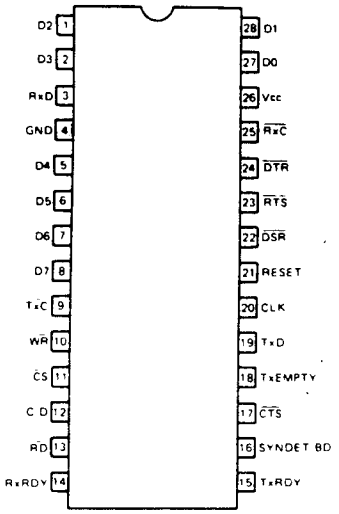
FEATURES

- Wide power supply voltage range from 3 V to 6 V.
- Wide temperature range from −40°C to 85°C.
- Synchronous communication upto 64K baud.
- Asynchronous communication upto 38.4K baud.
- Transmitting/receiving operations under double buffered configuration.
- Error detection (parity, overrun and framing)
- 28-pin DIP (MSM82C51ARS)
- 32-pin flat package (MSM82C51AGSK)

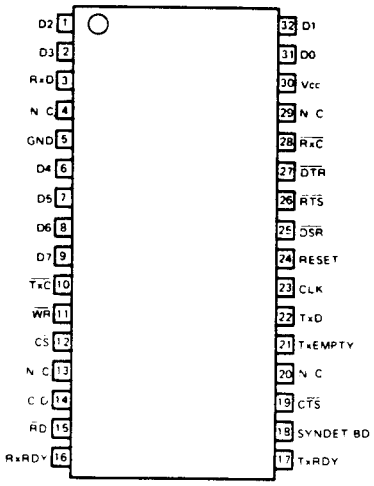
FUNCTIONAL BLOCK DIAGRAM



PIN CONFIGURATION



MSM82C51ARS (Top View)
28 Lead Plastic DIP



MSM82C51AGSK (Top View)
32 Lead Plastic Flat Package

FUNCTION

Outline
MSM82C51A's functional configuration is programmed by the software.

Operation between MSM82C51A and CPU is executed by program control. Table 1 shows the operation between CPU and the device.

Table 1 Operation between MSM82C51A and CPU

$\overline{CS}$	C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	
1	X	X	X	Data bus 3-state
0	X	1	1	Data bus 3-state
0	1	0	1	Status → CPU
0	1	1	0	Control word ← CPU
0	0	0	1	Data → CPU
0	0	1	0	Data ← CPU

It is necessary to execute a function-setting sequence after resetting on MSM82C51A. Fig. 1 shows the function-setting sequence.

If the function was set, the device is ready to receive a command, thus enabling the transfer of data by setting a necessary command, reading a status and reading/writing data.

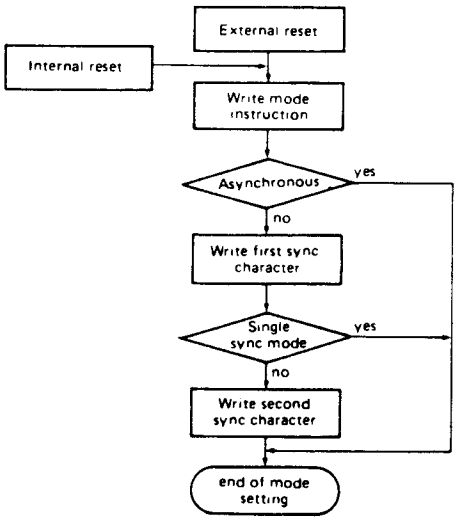


Fig. 1 Function-Setting Sequence (Mode Instruction Sequence)

Control Words
There are two types of control words

1. Mode instruction
2. Command

1. Mode Instruction
Mode instruction is used for setting the function of the MSM82C51A. Mode instruction will be in "wait for write" at either internal reset or external reset. Thus writing a control word after resetting will be recognized as "mode instruction."

Items to be set by mode instruction are as follows:

- Synchronous/asynchronous mode
- Stop bit length (asynchronous mode)
- Character length
- Parity bit
- Baud rate factor (asynchronous mode)
- Internal/external synchronization (synchronous mode)
- No. of synchronous characters (synchronous mode)

The bit configuration of the mode instruction is shown in Figs. 2 and 3. In the case of synchronous mode, it is necessary to write one- or two-sync characters.

If sync characters were written, a function will be set because the writing of sync characters constitutes part of the mode instruction.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S_2	S_1	EP	PEN	L_2	L_1	B_2	B_1

Baud rate factor

0	1	0	1
0	0	1	1
* Sync Mode	1x	16x	64x

Character length

0	1	0	1
0	0	1	1
5 bits	6 bits	7 bits	8 bits

Parity check

0	1	0	1
0	0	1	1
Disable	Odd parity	Disable	Even parity

Stop bit length

0	1	0	1
0	0	1	1
Inhibit	1 bit	1.5 bits	2 bits

*Refer to Figure 3

Fig. 2 Bit Configuration of Mode Instruction (Asynchronous)

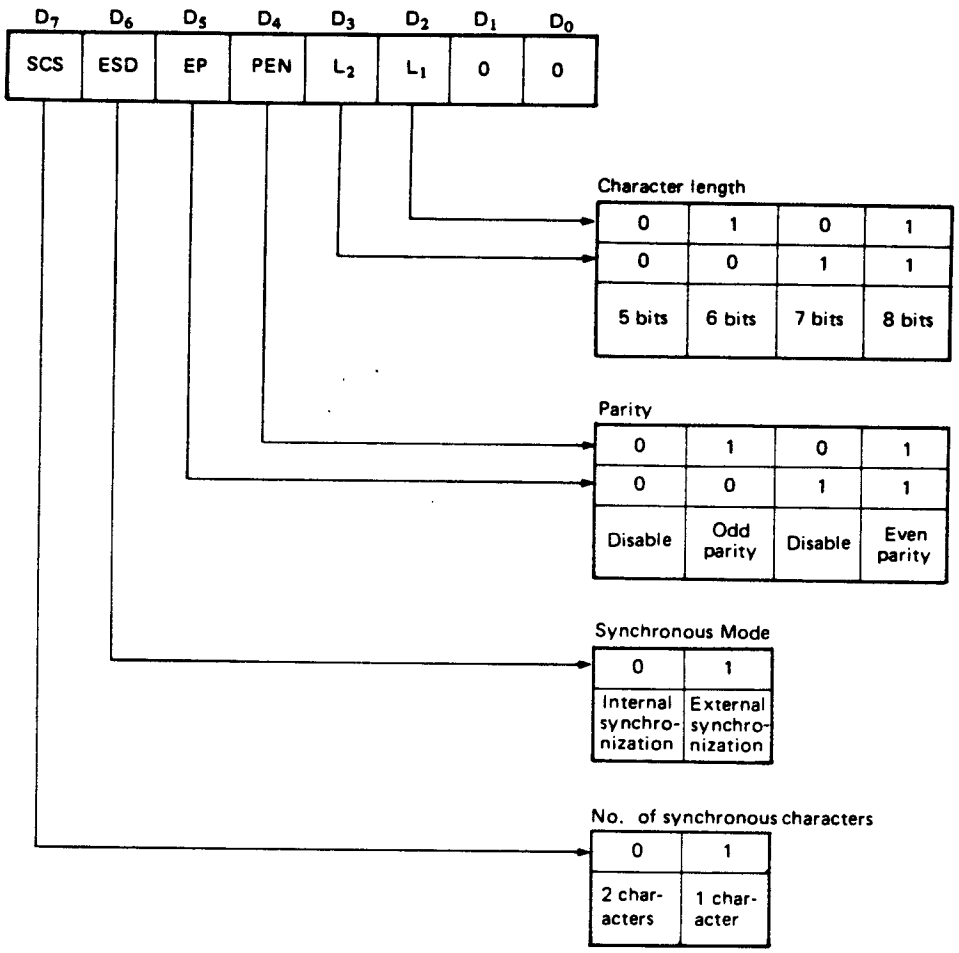


Fig. 3 Bit Configuration of Mode Instruction (Synchronous)

2. Command

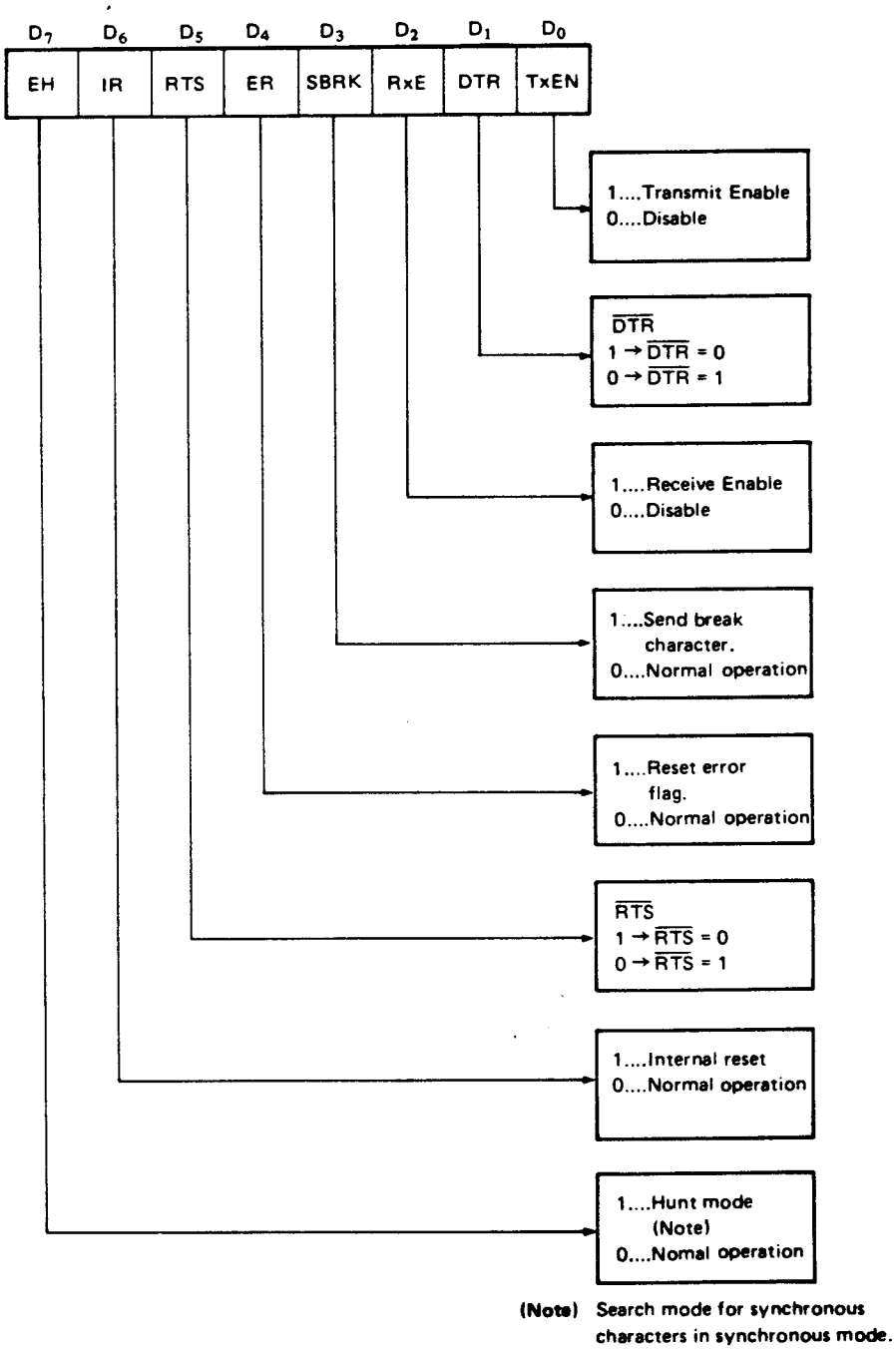
The command word is used for setting the operation of MSM82C51A.

It is possible to write a command whenever necessary after writing mode instruction and sync characters.

Items to be set by command are as follows:

- Transmit Enable/Disable
- Receive Enable/Disable
- $\overline{DTR}$, $\overline{RTS}$ Output of data.
- Resetting of error flag.
- Sending of break characters
- Internal reset
- Hunt mode (synchronous mode)

The bit configuration of a command is shown in Fig. 4.



(Note) Search mode for synchronous characters in synchronous mode.

Fig. 4 Bit Configuration of Command

Status Word

It is possible to see the internal status of MSM82C51A by reading a status word.

The bit configuration of status word is shown in Fig. 5.

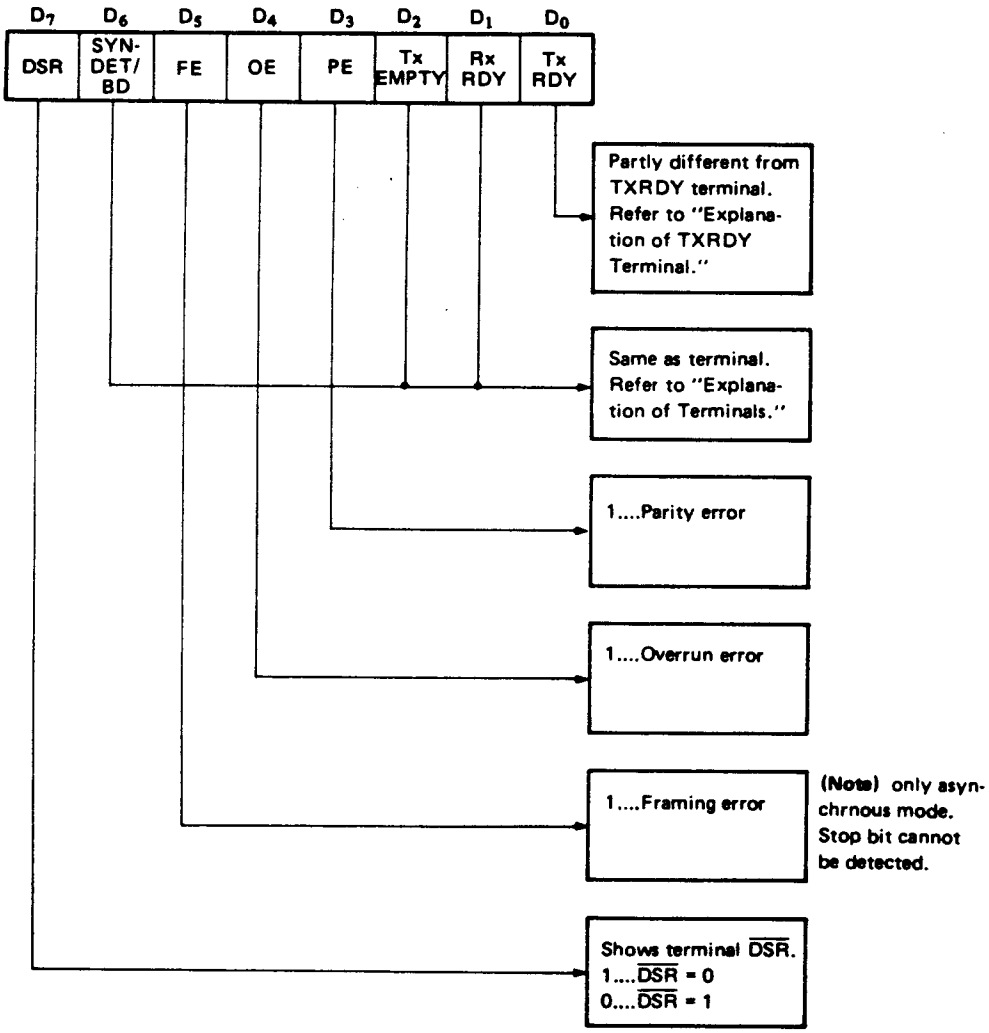


Fig. 5 Bit Configuration of Status Word

Standby Mode

It is possible to put MSM82C51A in "standby mode" for the complete static configuration of CMOS.

It is when the following conditions have been satisfied that MSM82C51A is in "standby mode."

(1) $\overline{CS}$ terminal shall be fixed at Vcc level.

(2) Input pins other than $\overline{CS}$, D0 to D7, $\overline{RD}$, $\overline{WR}$ and C/$\overline{D}$ shall be fixed at Vcc or GND level (including SYNDET in external synchronous mode).

Note: When all outputs current are low, ICCS specification applies.

Explanation of Each Terminal

D0 to D7 (I/O terminal)
This is a bidirectional data bus which receives control words and transmits data from the CPU and sends the status words and received data to the CPU.

RESET (Inpt terminal)
A "High" on this input forces the MSM82C51A into "reset."

The device waits for the "mode instruction."

The min. reset width is six clock inputs.

CLK (Input terminal)
CLK signal is used to generate internal device timing.

CLK signal is independent of $\overline{RXC}$ or $\overline{TXC}$.

However, the frequency of CLK must be greater than 30 times the $\overline{RXC}$ and $\overline{TXC}$ at Synchronous mode and Asynchronous "×1" mode, and must be greater than 5 times at Asynchronous "×16" and "×64" mode.

$\overline{WR}$ (Input terminal)
This is "active low" input terminal which receives a signal for writing transmit data and control words from CPU into the MSM82C51A.

$\overline{RD}$ (Input terminal)
This is "active low" input terminal which receives a signal for reading receive data and status words from the MSM82C51A.

C/$\overline{D}$ (Input terminal)
This is an input terminal which receives a signal for selecting data or command word and status word when MSM82C51A is accessed by CPU.

If C/$\overline{D}$ = low, data will be accessed.

If C/$\overline{D}$ = high, command word or status word will be accessed.

$\overline{CS}$ (Input terminal)
This is "active low" input terminal which selects the MSM82C51A.

Note The device won't be in "standby mode" only setting $\overline{CS}$ = High. Refer to "Explanation of Standby Mode."

TXD (Output terminal)
This is an output for serial transmit data.

The device is in "mark state" (high level) after resetting or when transmit is disabled.

It is also possible to set the device in the "break state" (low level) by a command.

TXRDY (Output terminal)
This is an output which indicates that the MSM82C51A is ready to accept a transmit data character. But the terminal is always at low level if $\overline{CTS}$ = high or the device was set in "TX disable status" by a command.

Note TXRDY of the status word indicates that transmit data character is receivable, regardless of $\overline{CTS}$ or command.

If the CPU writes a data character, TXRDY will be reset by the leading edge of the $\overline{WR}$ signal.

TXEMPTY (Output terminal)
This is an output terminal which indicates that the MSM82C51A transmitted all the characters and has no data characters to send.

In "synchronous mode," the terminal is at high level, if transmit data characters are no longer left (sync characters are automatically transmitted).

If the CPU writes a data character, TXEMPTY will be reset by the leading edge of $\overline{WR}$ signal.

Note As transmitter is disabled by setting CTS "High" or command, data written prior to the transmitter being disabled will be sent out, then TXD and TXEMPTY will be "High". If data is written after the transmitter is disabled, that data is not sent out and TXE will be "High". After re-enabling the transmitter it will be sent. (Refer to Transmitter Control and Flag Timing Chart.)

$\overline{TXC}$ (Input terminal)
This is a clock input signal which determines the transfer speed of transmit data.

In "synchronous mode," the baud rate will be the same as the frequency of $\overline{TXC}$.

In "Asynchronous mode," it is possible to select baud rate factor by the mode instruction.

It can be 1, 1/16, or 1/64 the $\overline{TXC}$.

The falling edge of $\overline{TXC}$ shifts the serial data out of the MSM82C51A.

RXD (Input terminal)
This is a terminal which receives serial data.

RXRDY (Output terminal)
This is a terminal which indicates that MSM82C51A contains a character that is ready to be read.

If CPU reads a data character, RXRDY will be reset by the leading edge of the $\overline{RD}$ signal.

Unless the CPU reads a data character before the next character is received completely, the preceding data will be lost. In such a case, the overrun error flag of the status register will be set.

$\overline{RXC}$ (Input terminal)
This is a clock input signal which determines the transfer speed of the receiver.

In "synchronous mode," the baud rate will be the same as the frequency of $\overline{RXC}$.

In "asynchronous mode," is is possible to select baud rate factor by mode instruction.

It can be 1, 1/16, 1/64 the $\overline{RXC}$.

SYNDET/BD (Input or output terminal)
This is a terminal whose function changes according to the mode.

In "internal synchronous mode," this terminal is at high level, if sync characters are received and synchronized. If status word is read, the terminal will be reset.

In "external synchronous mode," this is an input terminal.

A "High" on this input forces the MSM82C51A to start receiving data characters.

In "asynchronous mode," this is an output terminal which generates a high output upon the detection of a "break" character, if the receiver data contained "low-level" space between stop bits of two continuous characters. The terminal will be reset, if RXD is at high level.

$\overline{DSR}$ (Input terminal)
This is an input port for MODEM interfaces. The input status of the terminal can be read by reading the status register.

$\overline{DTR}$ (Output terminal)
This is an output port for MODEM interfaces. It is possible to set the status of $\overline{DTR}$ by a command.

$\overline{CTS}$ (Input terminal)
This is an input terminal for MODEM interfaces which is used for controlling the transmission. The terminal controls data transmission if the device is set in "TX Enable" status by a command. Data is transmittable if the terminal is at low level.

$\overline{RTS}$ (Output terminal)
This is an output port for MODEM interfaces. It is possible to set the status of $\overline{RTS}$ by a command.

ABSOLUTE MAXIMUM RATINGS

Parameter	Symbol	Limits		Unit	Conditions
		MSM82C51ARS	MSM82C51AGS		
Power supply voltage	V_{CC}	−0.5 ~ +7		V	With respect to GND
Input voltage	V_{IN}	−0.5 ~ V_{CC} + 0.5		V	
Output voltage	V_{OUT}	−0.5 ~ V_{CC} + 0.5		V	
Storage temperature	T_{stg}	−55 ~ 150		°C	—
Power dissipation	P_D	0.9	0.7	W	Ta = 25°C

OPERATING RANGE

Parameter	Symbol	Limits	Unit
Power supply voltage	V_{CC}	3 ~ 6	V
Operating temperature	T_{OP}	−40 ~ 85	°C

RECOMMENDED OPERATING CONDITIONS

Parameter	Symbol	Min.	Typ.	Max.	Unit
Power supply voltage	V_{CC}	4.5	5	5.5	V
Operating temperature	T_{OP}	−40	+25	+85	°C
"L" input voltage	V_{IL}	−0.3		+0.8	V
"H" input voltage	V_{IH}	2.2		V_{CC} + 0.3	V

DC CHARACTERISTICS

(V_{CC} = 4.5 ~ 5.5V Ta = −40°C ~ +85°C)

Parameter	Symbol	Min.	Typ.	Max.	Unit	Measurement Conditions
"L" output voltage	V_{OL}			0.45	V	I_{OL} = 2 mA
"H" output voltage	V_{OH}	3.7			V	I_{OH} = −400 μA
Input leakage current	I_{LI}	−10		10	μA	$0 \leq V_{IN} \leq V_{CC}$
Output leakage current	I_{LO}	−10		10	μA	$0 \leq V_{OUT} \leq V_{CC}$
Operating supply current	I_{CCO}			5	mA	Asynchronous X64 during transmitting/receiving
Standby supply current	I_{CCS}			100	μA	All input voltage shall be fixed at V_{CC} or GND level.

AC CHARACTERISTICS

(V_{CC} = 4.5 ~ 5.5V, Ta = −40 ~ 85°C)

CPU Bus Interface Part

Parameter	Symbol	Min.	Max.	Unit	Remarks
Address stable before $\overline{RD}$	t_{AR}	20		NS	Note 2
Address hold time for $\overline{RD}$	t_{RA}	20		NS	Note 2
$\overline{RD}$ pulse width	t_{RR}	250		NS	
Data delay from $\overline{RD}$	t_{RD}		200	NS	
$\overline{RD}$ to data float	t_{DF}	10	100	NS	
Recovery time between $\overline{RD}$	t_{RVR}	6		T_{cy}	Note 5
Address stable before $\overline{WR}$	t_{AW}	20		NS	Note 2
Address hold time for $\overline{WR}$	t_{WA}	20		NS	Note 2
$\overline{WR}$ pulse width	t_{WW}	250		NS	
Data set-up time for $\overline{WR}$	t_{DW}	150		NS	
Data hold time for $\overline{WR}$	t_{WD}	20		NS	
Recovery time between $\overline{WR}$	t_{RVW}	6		T_{cy}	Note 4
RESET pulse width	t_{RESW}	6		T_{cy}	

Serial Interface Part

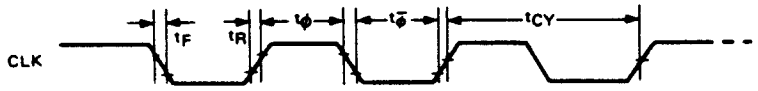
Parameter		Symbol	Min.	Max.	Unit	Remarks
Main clock period		t_{cy}	250		NS	Note 3
Clock low time		$t_{\overline{\phi}}$	90		NS	
Clock high time		t_{ϕ}	120	t_{cy}-90	NS	
Clock rise/fall time		t_R, t_F		20	NS	
TXD delay from falling edge of $\overline{TXC}$		t_{DTX}		1	μS	
Transmitter clock frequency	1X Baud	f_{TX}	DC	64	kHz	Note 3
	16X, Baud	f_{TX}	DC	615	kHz	
	64X, Baud	f_{TX}	DC	615	kHz	
Transmitter clock low time	1X Baud	t_{TPW}	13		T_{cy}	
	16X, 64X Baud	t_{TPW}	2		T_{cy}	
Transmitter clock high time	1X Baud	t_{TPD}	15		T_{cy}	
	16X, 64X Baud	t_{TPD}	3		T_{cy}	
Receiver clock frequency	1X Baud	f_{RX}	DC	64	kHz	Note 3
	16X Baud	f_{RX}	DC	615	kHz	
	64X Baud	f_{RX}	DC	615	kHz	
Receiver clock low time	1X Baud	t_{RPW}	13		T_{cy}	
	16X, 64X Baud	t_{RPW}	2		T_{cy}	
Receiver clock high time	1X Baud	t_{RPD}	15		T_{cy}	
	16X, 64X Baud	t_{RPD}	3		T_{cy}	
Time from the center of last bit to the rise of TXRDY		t_{TXRDY}		8	T_{cy}	
Time from the leading edge of $\overline{WR}$ to the fall of TXRDY		$t_{TXRDY\ CLEAR}$		400	NS	
Time from the center of last bit to the rise of RXRDY		t_{RXRDY}		26	T_{cy}	

Parameter	Symbol	Min.	Max.	Unit	Remarks
Time from the leading edge of $\overline{RD}$ to the fall of RXRDY	$t_{RXRDY\ CLEAR}$		400	NS	
Internal SYNDET delay time from rising edge of $\overline{RXC}$	t_{IS}		26	T_{cy}	
SYNDET setup time for $\overline{RXC}$	t_{ES}	18		T_{cy}	
TXE delay time from the center of last bit	$t_{TXEMPTY}$	20		T_{cy}	
MODEM control signal delay time from rising edge of $\overline{WR}$	t_{WC}	8		T_{cy}	
MODEM control signal setup time for falling edge of $\overline{RD}$	t_{CR}	20		T_{cy}	
RXD setup time for rising edge of $\overline{RXC}$ (1X Baud)	t_{RXDS}	11		T_{cy}	
RXD hold time for falling edge of $\overline{RXC}$ (1X Baud)	t_{RXDH}	17		T_{cy}	

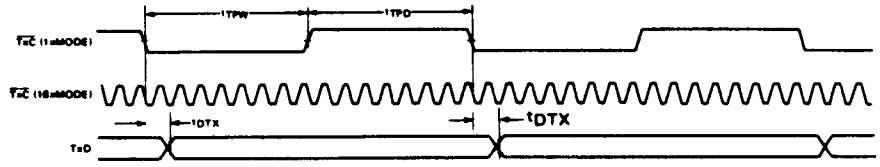
Caution 1) AC characteristics are measured at 150 pF capacity load as an output load based on 0.8V at low level and 2.2 V at high level for output and 1.5 V for input.
2) Addresses are $\overline{CS}$ and C/$\overline{D}$.
3) f_{TX} or $f_{RX} \leq 1/(30\ Tcy)$ 1 × baud
f_{TX} or $f_{RX} \leq (1/5\ Tcy)$ 16 ×, 64 × Baud
4) This recovery time is mode initialization only. Recovery time between command writes for Asynchronous Mode is 8 t_{CY} and for Synchronous Mode is 18 t_{CY}.
Write Data is allowed only when TXRDY = 1.
5) This recovery time is Status read only.
Read Data is allowed only when RXRDY = 1.
6) Status update can have a maximum delay of 28 clock periods from event affecting the status.

TIMING CHART

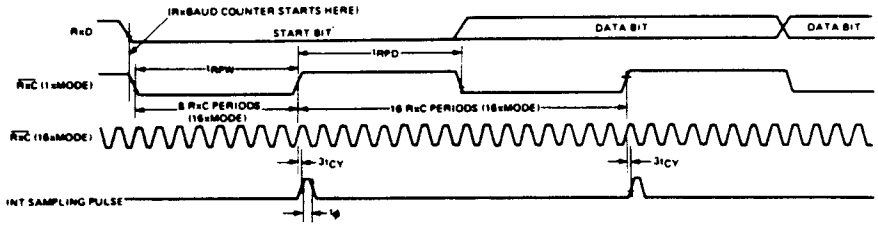
System Clock Input



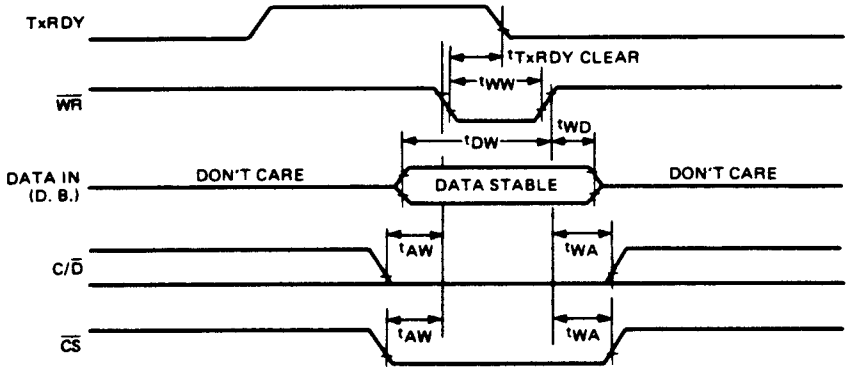
Transmitter Clock and Data



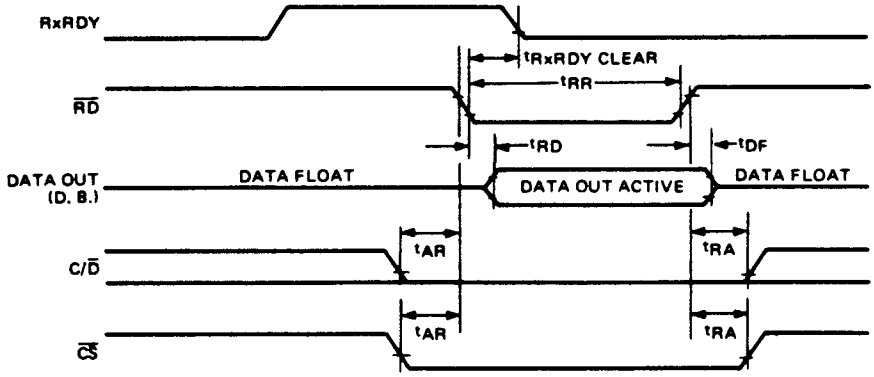
Receiver Clock and Data



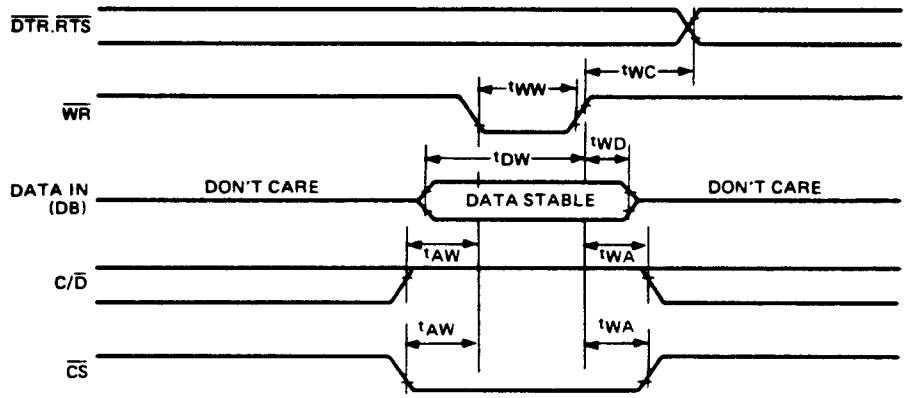
Write Data Cycle (CPU → USART)



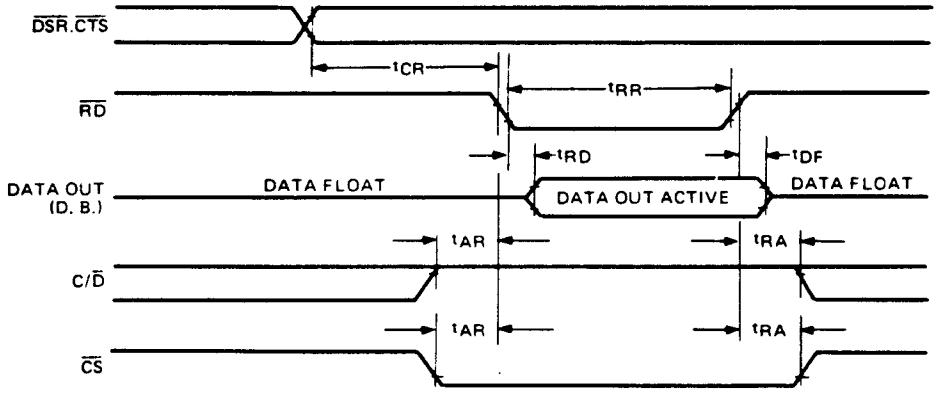
Read Data Cycle (CPU ← USART)



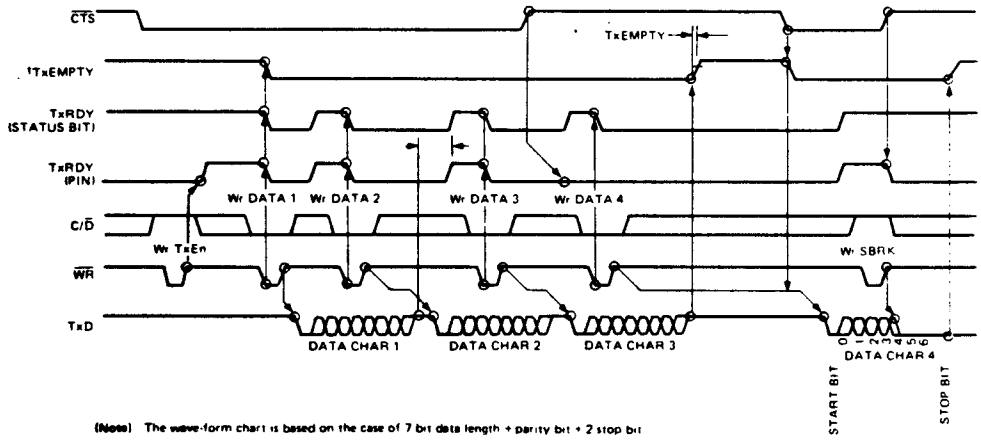
Write Control or Output Port Cycle (CPU → USART)



Read Control or Input Port (CPU ← USART)

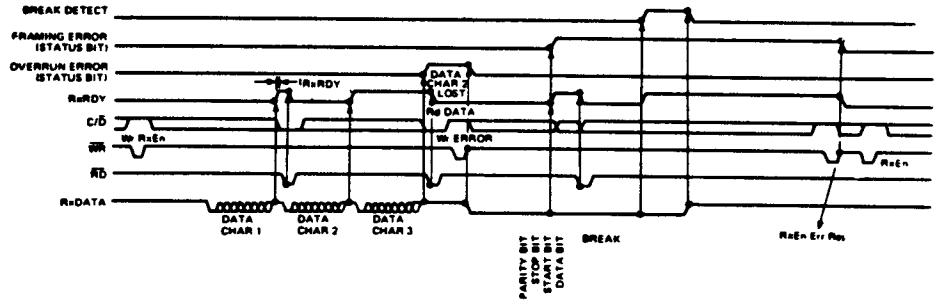


Transmitter Control and Flag Timing (ASYNC Mode)



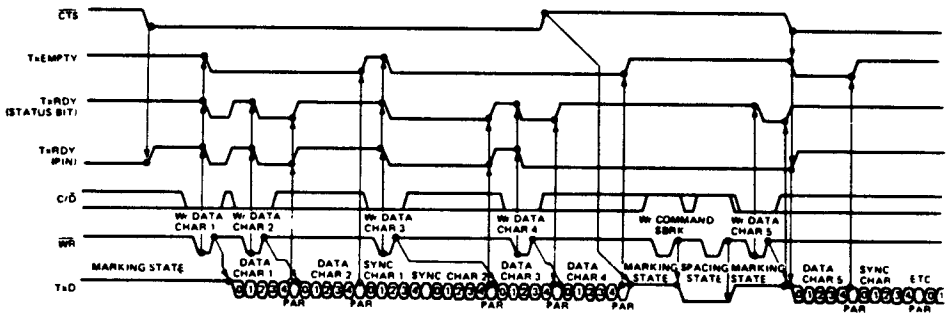
(Note) The wave-form chart is based on the case of 7 bit data length + parity bit + 2 stop bit

Receiver Control and Flag Timing (ASYNC Mode)



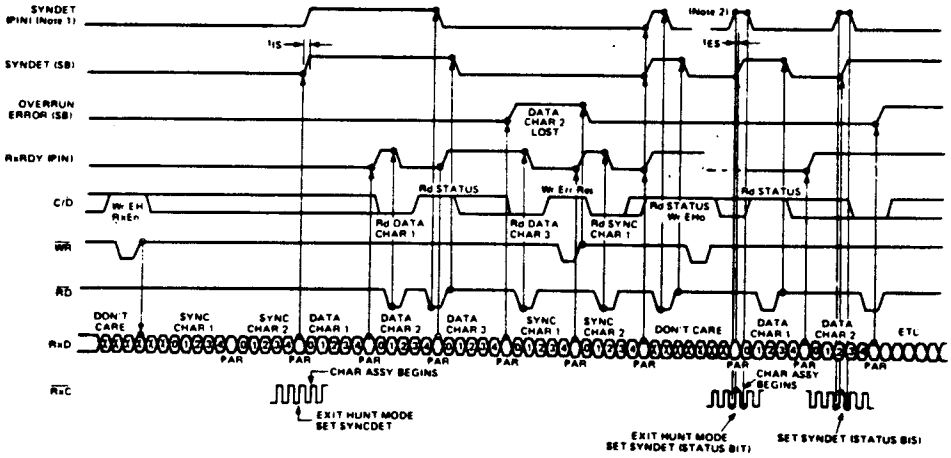
(Note) The wave-form chart is based on the case of 7 data bit length + parity bit + 2 stop bit

Transmitter Control and Flag Timing (SYNC Mode)



(Note) The wave-form chart is based on the case of 5 data bit length + parity bit and 2 synchronous characters

Receiver Control and Flag Timing (SYNC Mode)



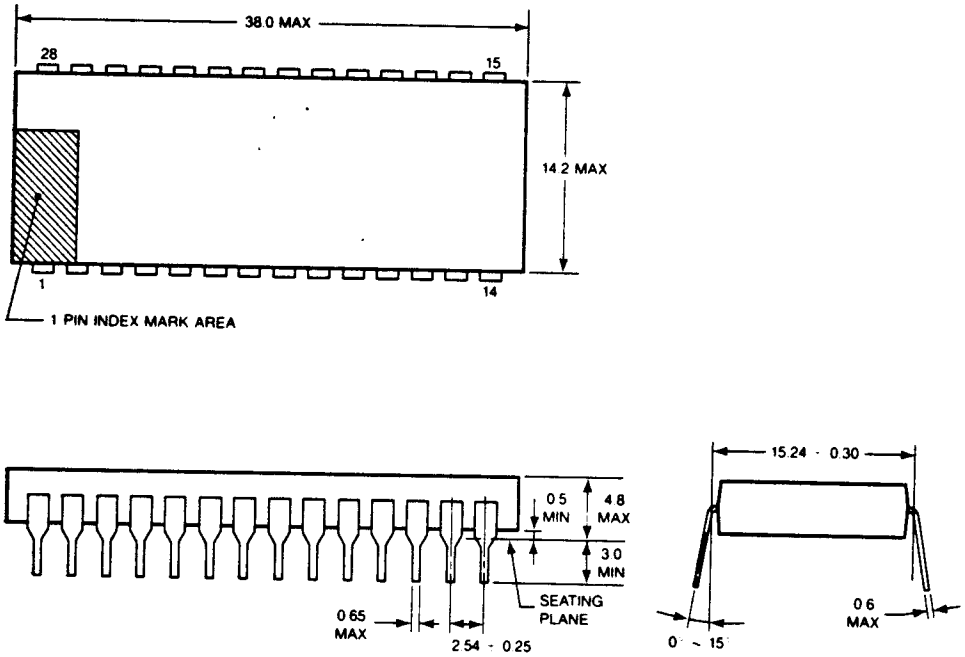
(Note 1) Internal synchronization is based on the case of 5 data bit length + parity bit and 2 synchronous characters

(Note 2) External synchronization is based on the case of 5 data bit length + parity bit

PACKAGE SPECIFICATIONS

MSM82C51ARS
28 LEAD PLASTIC DIP

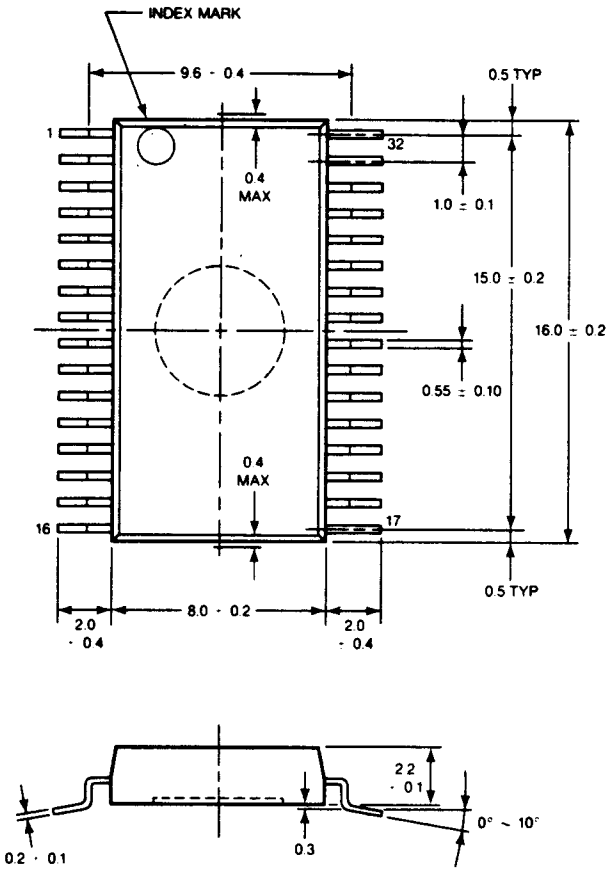
(UNIT: mm)



PACKAGE SPECIFICATIONS cont'd

MSM82C51AGSK
32 LEAD PLASTIC FLAT PACKAGE

(UNIT: mm)



OKI SEMICONDUCTOR, INC. 650 N. MARY AVENUE, SUNNYVALE, CA 94086

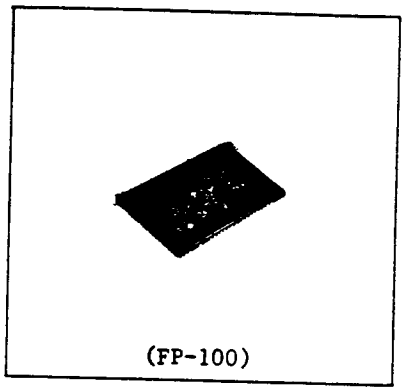
TELEPHONE: (408) 720-1900 TELEX (25) 910-3380508

PRINTED IN U.S.A.

Hitachi HD61102A LCD Column Driver Data Sheet

HD61102 (DOT MATRIX LIQUID CRYSTAL GRAPHIC DISPLAY COMMON DRIVER)

HD61102 is a column (segment) driver for dot matrix liquid crystal graphic display systems. It stores the display data transferred from a 8-bit micro-computer in the internal display RAM and generates dot matrix liquid crystal driving signals.



(FP-100)

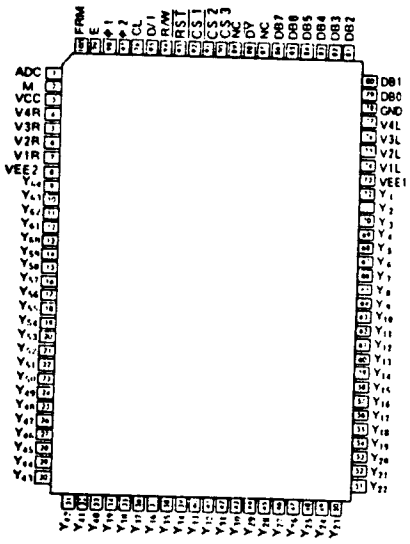
Each bit data of display RAM corresponds to ON/OFF of each dot of liquid crystal display to provide more flexible display.

As it is internally equipped with 64 output drivers for display, it is available for liquid crystal graphic display with many dots.

The HD61102, which is produced in the CMOS process, can accomplish a portable battery drive equipment by combining a CMOS micro-computer, utilizing the liquid crystal display's lower power dissipation.

Moreover it can facilitate dot matrix liquid crystal graphic display system configuration by combining the row (common) driver HD61103A.

■ PIN ARRANGEMENT



(Top view)

■ FEATURES

- Dot matrix liquid crystal graphic display column driver incorporating display RAM.
- RAM data direct display by internal display RAM
  RAM bit data "1" ..... ON
  RAM bit data "0" ..... OFF
- Internal display RAM address counter
  preset, increment
- Display RAM capacity ..... 512 bytes (4096 bits)
- 8-bit parallel interface
- Internal liquid crystal display driver circuit 64
- Display duty
  Combination of frame control signal and data latch synchronization signal make it possible to select out of static through an optional duty.
- Wide range of instruction function
  Display Data Read/Write, Display ON/OFF,
  Set address, Set Display Start line,
  Read Status
- Lower power dissipation ——— during display 2mW max
- Power supply Vcc ——— +5V ± 10%
  VEE ——— 0V ∿ -10V
- Liquid crystal display driving level —— 15.5V max
- CMOS process
- 100 - pin flat plastic package (FP-100)

■ ABSOLUTE MAXIMUM RATINGS

Item	Symbol	Value	Unit	Note
Supply voltage	V_{CC}	-0.3 ∿ +7.0	V	2
	V_{EE}	V_{CC} -16.5 ∿ V_{CC} +0.3	V	3
Terminal voltage (1)	V_{T1}	V_{EE} -0.3 ∿ V_{CC} +0.3	V	4
Terminal voltage (2)	V_{T2}	-0.3 ∿ V_{CC} +0.3	V	2, 5
Operating temperature	Topr	-20 ∿ +75	°C	
Storage temperature	Tstg	-55 ∿ +125	°C	

(Note 1) LSI's may be destroyed for ever, if being used beyond the absolute maximum ratings.
In ordinary operation, it is desirable to use them observing the recommended operation conditions.
Using beyond these conditions may cause malfunction and poor reliability.

(Note 2) All voltage values are referred to GND=0V.

(Note 3) Apply the same supply voltage to V_{EE} 1 and V_{EE}2.

(Note 4) Applies to V1L, V2L, V3L, V4L, V1R, V2R, V3R and V4R.
Maintain

$$V_{CC} \geq V1L = V1R \geq V3L = V3R \geq V4L = V4R \geq V2L = V2R \geq V_{EE}$$

(Note 5) Applies to M, FRM, CL, $\overline{RST}$, ADC, ϕ1, ϕ2, $\overline{CS1}$, $\overline{CS2}$, CS3, E, R/W, D/I, ADC and DB0∿7.

■ ELECTRICAL CHARACTERISTICS
(GND=0V, VCC=4.5 ∿ 5.5V, VEE=0∿-10V, Ta=-20∿+75°C)

Item	Symbol	Test condition	Limit min	Limit typ	Limit max	Unit
Input "High" voltage	V_{IHC}		0.7×Vcc	-	Vcc	V
	V_{IHT}		2.0	-	Vcc	V
Input "Low" voltage	V_{ILC}		0	-	0.3×Vcc	V
	V_{ILT}		0	-	0.8	V
Output "High" voltage	V_{OH}	IOH=-205μA	2.4	-	-	V
Output "Low" voltage	V_{OL}	IOL=1.6mA	-	-	0.4	V
Input leakage current	I_{IL}	Vin=GND∿Vcc	-1.0	-	+1.0	μA
Three state (OFF) input current	I_{TSL}	Vin=GND∿Vcc	-5.0	-	+5.0	μA
Liquid crystal supply leakage current	I_{LSL}	Vin=VEE∿Vcc	-2.0	-	+2.0	μA
Driver ON resistance	R_{ON}	Vcc-VEE=15V ±I_{LOAD}=0.1mA	-	-	7.5	KΩ
Dissipation current	Icc(1)	During display	-	-	100	μA
	Icc(2)	During access access cycle=1MHz	-	-	500	μA

(Note 1) Applies to M, FRM, CL, $\overline{RST}$, ADC, ADC, ϕ1 and ϕ2.

(Note 2) Applies to $\overline{CS1}$, $\overline{CS2}$, CS3, E, R/W, D/I and DB0 ∿ 7.

(Note 3) Applies to DB0 ∿ 7.

(Note 4) Applies to terminals except for DB0 ∿ 7.

(Note 5) Applies to DB0 ∿ 7 at high impedance.

(Note 6) Applies to V1L ∿ V4L and V1R ∿ V4R.

(Note 7) Applies to Y1 ∿ Y64.

(Note 8) Specified when liquid crystal display is in 1/64 duty.

Operation frequency f_{CLK}=250 kHz (ϕ1 and ϕ2 frequency)

Frame frequency f_M =70 Hz (FRM frequency)

Specified in the state of

Output terminal ----- not loaded

Input level --------- V_{IH}=Vcc(V)

V_{IL}=GND (V)

Measured at Vcc terminal

● INTERFACE AC CHARACTERISTICS

(1) MPU Interface

(GND=0V, Vcc=4.5 ∿ 5.5V, V_{EE}=0 ∿ -10V, Ta=-20 ∿ +75°C)

Item	Symbol	min	typ	max	Unit	Note
E cycle time	t_{CYC}	1000	-	-	ns	1, 2
E high level width	P_{WEH}	450	-	-	ns	1, 2
E low level width	P_{WEL}	450	-	-	ns	1, 2
E rise time	tr	-	-	25	ns	1, 2
E fall time	tf	-	-	25	ns	1, 2
Address setup time	t_{AS}	140	-	-	ns	1, 2
Address hold time	t_{AH}	10	-	-	ns	1, 2
Data setup time	t_{DSW}	200	-	-	ns	1
Data delay time	t_{DDR}	-	-	320	ns	2, 3
Data hold time (Write)	t_{DHW}	10	-	-	ns	1
Data hold time (Read)	t_{DHR}	20	-	-	ns	2

(Note 1)

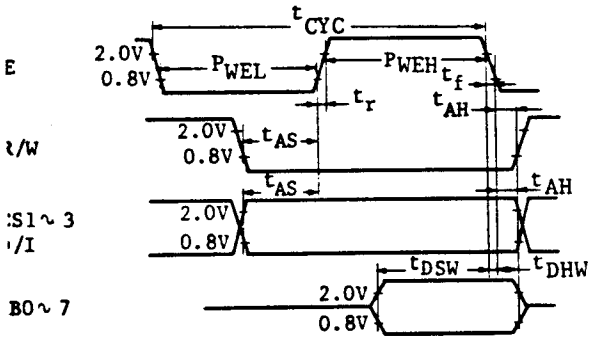


Fig. 1 CPU Write Timing

(Note 2)

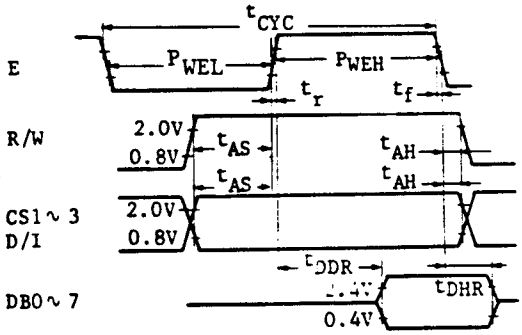
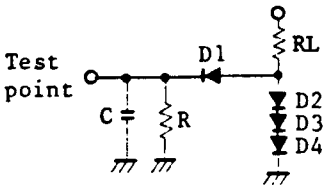


Fig. 2 CPU Read Timing

(Note 3) DB0 ∿ 7 : load circuit



RL=2.4KΩ

R =11KΩ

C =130pF (including jig capacity)

Diodes D1 to D4 are all 1S2074Ⓗ.

(2) Clock Timing

(GND=0V, Vcc=4.5 ∿ 5.5V, V_{EE}=0 ∿ -10V, Ta=20 ∿ +75°C)

Item	Symbol	Test condition	Limit			Unit
			min	typ	max	
φ1, φ2 cycle time	tcyc	Fig. 3	2.5	-	20	μs
φ1 "Low" level width	$t_{WL\phi1}$	Fig. 3	625	-	-	ns
φ2 "Low" level width	$t_{WL\phi2}$	Fig. 3	625	-	-	ns
φ1 "High" level width	$t_{WH\phi1}$	Fig. 3	1875	-	-	ns
φ2 "High" level width	$t_{WH\phi2}$	Fig. 3	1875	-	-	ns
φ1-φ2 phase difference	tD12	Fig. 3	625	-	-	ns
φ2-φ1 phase difference	tD21	Fig. 3	625	-	-	ns
φ1, φ2 rise time	tr	Fig. 3	-	-	150	ns
φ1, φ2 fall time	tf	Fig. 3	-	-	150	ns

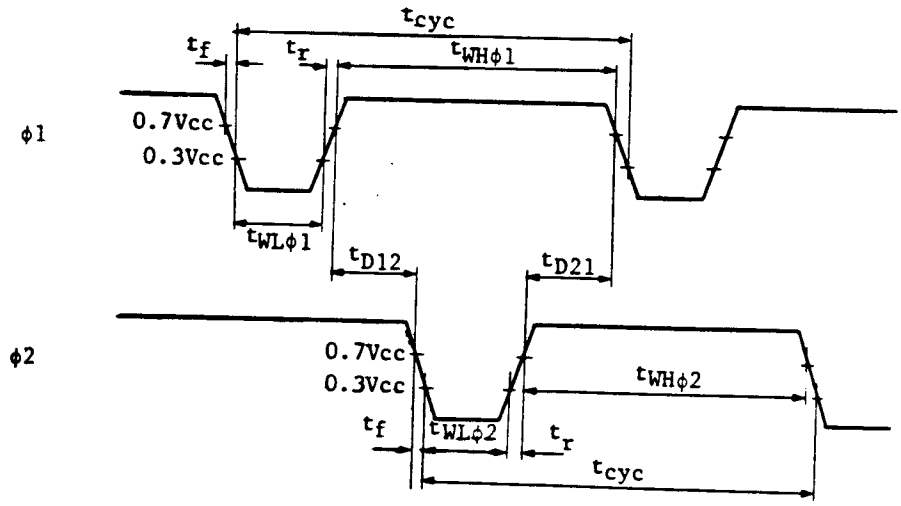


Fig. 3 External Clock Waveform

(3) Display Control Timing

(GND=0V, V_{CC}=4.5 ∿ 5.5V, V_{EE}=0 ∿ -10V, Ta=-20 ∿ +75°C)

Item	Symbol	Test condition	Limit			Unit
			min	typ	max	
FRM delay time	t_{DFRM}	Fig. 4	-2	-	+2	μs
M delay time	t_{DM}	Fig. 4	-2	-	+2	μs
CL "Low" level width	t_{WLCL}	Fig. 4	35	-	-	μs
CL "High" level width	t_{WHCL}	Fig. 4	35	-	-	μs

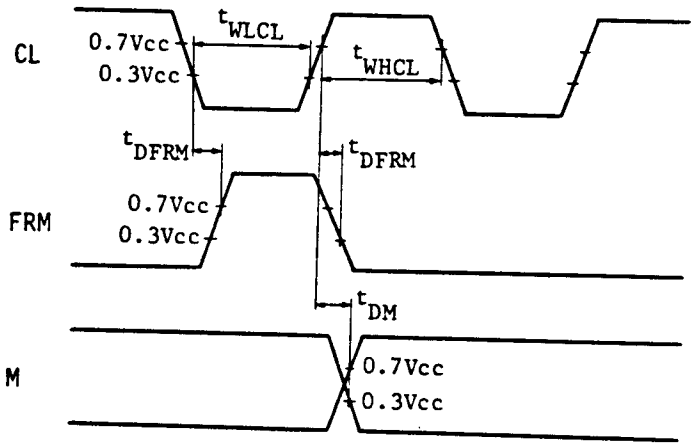
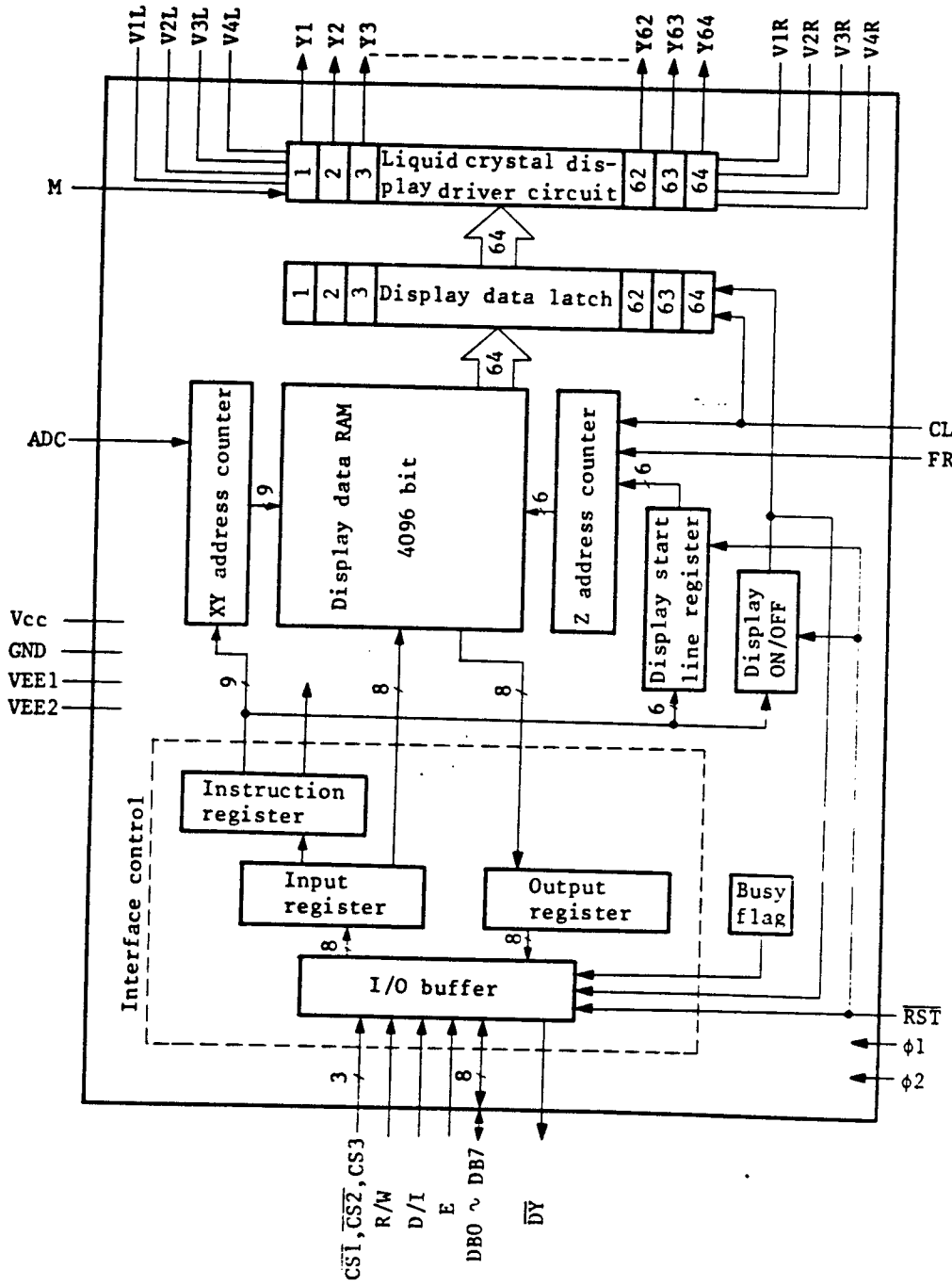


Fig. 4 Display Control Signal Waveform

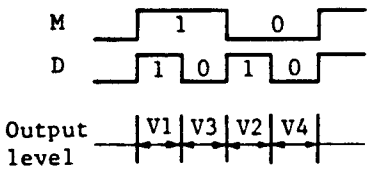
■ BLOCK DIAGRAM



■ TERMINAL FUNCTIONS

Terminal name	Number of terminals	I/O	Connected to	Functions
V_{CC} GND	2		Power supply	Power supply for internal logic. Recommended voltage is GND = 0V V_{CC} = +5V ± 10%
V_{EE} 1 V_{EE} 2	2		Power supply	Power supply for liquid crystal display drive circuit. Recommended power supply voltage is V_{CC} - 15 to GND. Connect the same power supply to V_{EE1} and V_{EE2}. V_{EE1} and V_{EE2} are not connected each other in the LSI.
V1L, V1R V2L, V2R V3L, V3R V4L, V4R	8		Power supply	Power supply for liquid crystal display drive. Apply the voltage specified depending on liquid crystals within the limit of V_{EE} through V_{CC}. V1L(V1R), V2L(V2R)---Selection level V3L(V3R),V4L(V4R)----Non-selection level Power supplies connected with V1L and V1R (V2L & V2R, V3L & V3R, V4L & V4R) should have the same voltages.
$\overline{CS1}$ $\overline{CS2}$ CS3	3	I	MPU	Chip selection. Data can be input or output when the terminals are in the next conditions. Terminal name: $\overline{CS1}$, $\overline{CS2}$, CS3 Condition: 'L', 'L', 'H'
E	1	I	MPU	Enable At write(R/W=L) : Data of DB0 to D[illegible] latched at the [illegible] E. At read(R/W=H) : Data appears at DB0 to DB7 while E is in "High" level.

Terminal name	Number of terminals	I/O	Connected to	Functions
R/W	1	I	MPU	Read/Write R/W=H : Data appears at DB0 to DB7 and can be read by the CPU When E=H, $\overline{CS1}$, $\overline{CS2}$=L and CS3=H. R/W=L : DB0 to DB7 can accept at fall of E when $\overline{CS1}$, $\overline{CS2}$=L and CS3=H.
D/I	1	I	MPU	Data/Instruction D/I=H : Indicates that the data of DB0 to DB7 is display data. D/I=L : Indicates that the data of DB0 to DB7 is display control data.
ADC	1	I	V_{CC}/GND	Adress control signal determine the relation between Y address of display RAM and terminals from which the data is output. ADC=H : Y1-$0, Y64-$63 ADC=L : Y64-$0, Y1-$63
DB0~DB7	8	I/O	MPU	Data bus, three-state I/O common terminal
M	1	I	HD61103A	Switch signal to convert liquid crystal drive waveform into AC.
FRM	1	I	HD61103A	Display synchronous signal (frame signal) This signal presets the 6-bit display line counter and synchronizes a common signal with the frame timing when the FRM signal becomes high.
CL	1	I	HD61103A	Syncronous signal to latch display data. The CL signal indicates to count up the display output adress counter and latch the display data at rising.
Φ1,Φ2	1	I	HD61103A	2-phase clock signal for internal operation. The Φ1 and Φ2 clocks are used to perform the operations (I/O of display data and execution of instructions) other than display.

Terminal name	Number of terminals	I/O	Connected to	Functions
Y1~Y64	64	O	Liquid criptal display	Liquid crystal display column (segment) drive output. These pins outputs light ON level when "1" is in the display RAM, and light OFF level with "0" in it. Relation among output level, M and display data (D) is as follows. 
$\overline{RST}$	1	I	CPU or external CR	The following registers can be initialized by setting the $\overline{RST}$ signal to "Low" level. (1) ON/OFF register 0 set (display OFF) (2) Display start line register 0 line set (displays from 0 line) After releasing reset, this condition can be changed only by the instruction.
$\overline{DY}$	1	O	Open	Output terminal for test. Usually, don't connect any lines to this terminal.
NC	2		Open	Unused terminals. Don't connect any lines to these terminals.

(Note) "1" corresponds to "High level" in positive logic.

■ FUNCTION OF EACH BLOCK

• Interface Control

(1) I/O buffer

Data is transferred through 8 data buses (DB0 ∿ DB7).

DB7 .... MSB (Most Significant Bit)

DB0 .... LSB (Least Significant Bit)

Data can neither be input nor output unless CS1 to CS3 are in the active mode. Therefore, when CS1 to CS3 are not in active mode it i useless to switch the signals of input terminals except $\overline{RST}$ and ADC, namely, the internal state is maintained and no instruction excute. Besides, pay attention to $\overline{RST}$ and ADC which operate irrespectively b CS1 to CS3.

(2) Register

Both input register and output register are provided to interface MPU of which the speed is different from that of internal operation. The selection of these registers depend on the combination of R/I an D/I signals.

Table 1. Register Selection

D/I	R/W	Operation
1	1	Reads data out of output register as internal operation (display data RAM → output register)
1	0	Writes data into input register as internal operation (input register → display data RAM)
0	1	Busy check. Read of status data.
0	0	Instruction

① Input register

Input register is used to store data temporarily before writing it int display data RAM.

The data from MPU is written into input register, then into display RAM automatically by internal operation.

When CS1 to CS3 are in the active mode and D/I and R/W select the input register as shown in Table 1, data is latched at the fall of E signal.

② Output register

Output register is used to store data temporarily which is read from display data RAM. To read out the data from output register, CS1 to CS3 should be in the active mode and both D/I and R/W should be 1. With READ instruction, data stored in the output register is output while E is "H" level. Then, at the fall of E, the display data at the indicated address is latched into the output register and the address is increased by 1. The contents in the output register is rewritten with READ instruction, while is held with address set instruction, etc.

Therefore, the data of the specified address can not be output with READ instruction soon after the address is set, but can be output at the second read of data. That is to say, one dummy read is necessary. Fig. 5 shows the CPU read timing.

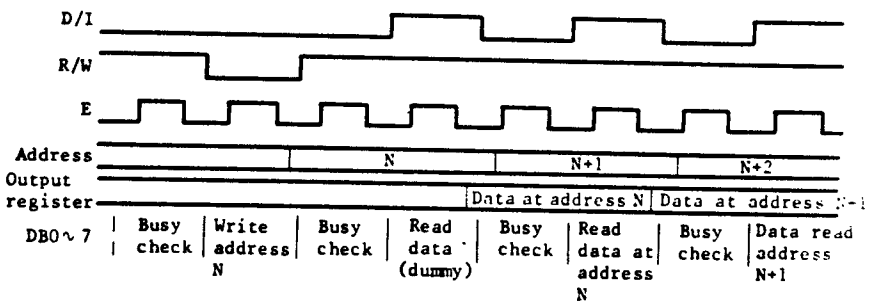
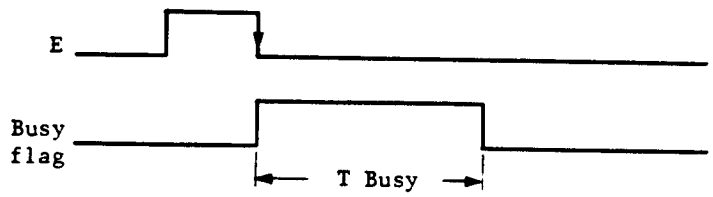


Fig. 5 CPU Read Timing

- Busy Flag

"1" of busy flag indicates that HD61102 is on the move and any instruction except Status Read instruction can not be accepted. The value of the busy flag is read out on DB7 by the Status Read instruction. Make sure that the busy flag is reset ("0") before the issue of instruction.



$1/f_{CLK} \leq T\ Busy \leq 3/f_{CLK}$

f_{CLK} is $\phi 1$, $\phi 2$ frequency

- Display ON/OFF Flip Flop

Display ON/OFF flip flop selects one of two states, ON state and OFF state of segments Y1 to Y64. In ON state, the display data corresponding to that in RAM is output to the segments. On the other hand, the display data at all segments disappear in OFF state independent of the data in RAM.
It is controlled by display ON/OFF instruction. '0' of $\overline{RST}$ signal sets the segments in OFF state. The status of the flip flop is output to DB5 by Status Read instruction. Display ON/OFF instruction does not influence data in RAM. To control display data latch by this flip flop, CL signal (display synchronous signal) should be input correctly.

- Display Start Line Register

The register specifies a line in RAM which corresponds to the top line of LCD panel, when displaying contents in display data RAM on the LCD panel. It is used for scrolling of the screen.
6-bit display start line information is written into this register by displa start line set instruction, with 'H' level of FRM signal instructing to st the display, the information in this register is transferred to Z address counter which controls the display address, and the Z address counter is preset.

- X, Y Address Counter

This is a 9-bit counter which designates addresses of internal display data RAM. X address counter of upper 3 bits and Y address counter of lower 6 bits should be set each address by respective instruction.

(1) X address counter

Ordinary register with no count functions. An address is set in by instructions.

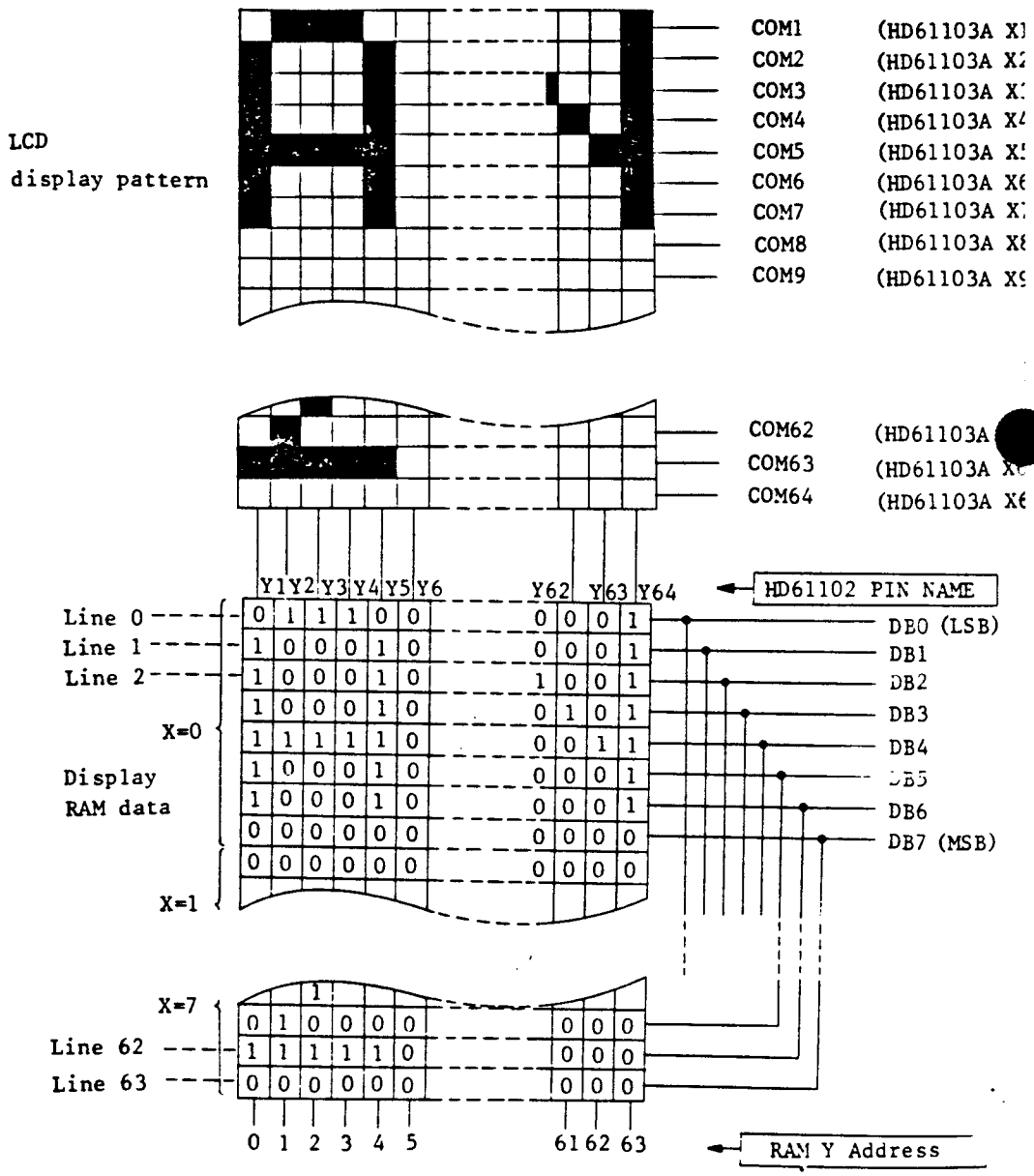
(2) Y address counter

An address is set in by instruction and it is increased by 1 automatically by R/W operations of display data. The Y address counter loops the values of 0 to 63 to count.

- Display Data RAM

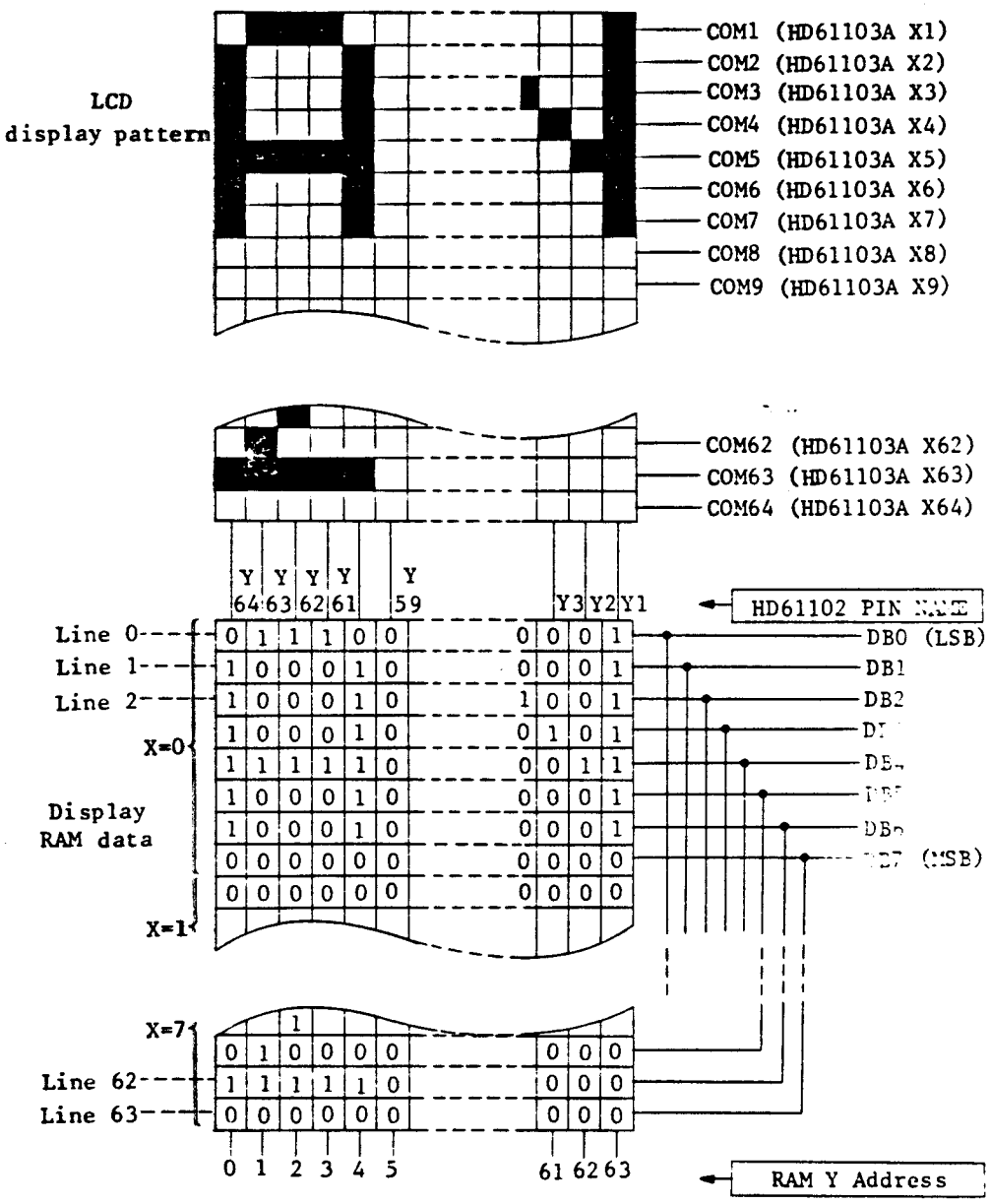
Dot data for display is stored in this RAM. 1-bit data of this RAM corresponds to light ON (data=1) and light OFF (data=0) of 1 dot in the display panel. The correspondence between Y addresses of RAM and segment PINs can be reversed by ADC signal.
As ADC signal controls Y address counter, a reverse of the signal during the operation causes malfunction and destruction of the contents of register and data of RAM. Therefore, never fail to connect ADC pin to V_{CC} or GND when using.

Fig. 6 shows the relations between Y address of RAM and segment pins in the cases of ADC=1 and ADC=0. (display start line=0, 1/64 duty).



(a) ADC="1" (Connected to Vcc)

Fig. 6 Relation between RAM Data and Display



(b) ADC="0" (Connected to GND)

Fig. 6 Relation Between RAM Data and Display

- Z Address Counter

The Z address counter generates addresses for outputting the display data synchronized with the common signal. This counter consists of 6-bit and counts up at the fall of CL signal. With "H" level of FRM, the contents of the display start line register is preset at the Z counter.

- Display data Latch

The display data latch stores the display data temporarily which is output from display data RAM to liquid crystal driving circuit.
Data is latched at the rise of CL signal. Display ON/OFF instruction controls the data in this latch and does not influence data in display data RAM.

- Liquid Crystal Display Driver Circuit

The combination of latched display data and M signal causes one of the 4 liquid crystal driver levels, V1, V2, V3 and V4 to be output.

- Reset

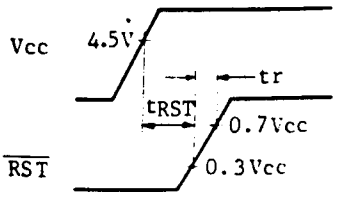
The system can be initialized by setting $\overline{RST}$ terminal at "Low" level when turning power ON.

1) Display-OFF
2) Set display start line register 0 line.

While $\overline{RST}$ is in Low level, any instruction except Status Read cannot be accepted. Therefore, Carry out other instructions after making sure that DB4=0 (clear RESET) and DB7=0 (Ready) by Status Read instruction.
The conditions of Power Supply at initial power up are as follows.

Item	Symbol	Min.	Typ	Max.	Unit
Reset time	t_{RST}	1.0	-	-	μs
Rise time	t_r	-	-	200	ns

Do not fail to set the system again because RESET during operation may destroy the data in all the register except ON/OFF register and in RAM.



■ DISPLAY CONTROL INSTRUCTIONS

• Outline

Table 2 shows the instructions. Read/Write (R/W) signal, Data/Instruction (D/I) signal and Data bus signal (DB0 to DB7) are also called instructions because the internal operation depends on the signals from MPU.
These explanations are detailed from the following page. Generally, there are following three kinds of instructions.

(1) Instruction to give addresses in the internal RAM
(2) Instruction to transfer data from/to the internal RAM
(3) Other instructions

In general use, the instruction (2) are used most frequently. But, since Y address of the internal RAM is increased by 1 automatically after writing (reading) data, the program can be lessened. During the execution of an instruction, the system cannot accept other instructions than Status Read instruction. Send instructions from MPU after making sure if the busy flag is "0", which is the proof an instruction is not being excuted.

Table 2. Instructions

Instructions	Code										Functions	
	R/W	D/1	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
1 Display ON/OFF	0	0	0	0	1	1	1	1	1	1/0	Controls the ON/OFF of display. RAM data and internal status are not affected. 1:ON, 0:OFF.	
2 Display start line	0	0	1	1	display start line (0∿63)						Specifies a RAM line displayed at the top of the screen.	
3 Set page (X address)	0	0	1	0	1	1	1	Page (0∿7)			Sets the page (X address) of RAM at the page (X address) register.	
4 Set Y address	0	0	0	1	Y address (0∿63)						Sets the Y address at the Y address counter	
5 Status Read	1	0	Busy	0	ON/OFF	RESET	0	0	0	0	Reads the status. RESET 1: reset 0:normal ON/OFF 1: display OFF 0:display ON Busy 1: on the internal operation 0: Ready	
6 Write display data	0	1	Write Data								Writes data DB0 (LSB) to DB7 (MSB) on the data bus into display RAM.	Has access to the address of the display RAM specified in advance. After the access, Y address is increased by 1.
7 Read display data	1	1	Read Data								Reads data DB0 (LSB) to DB7 (MSB) from the display RAM to the data bus.	

Note 1) Busy time varies with the frequency (f_{CLK}) of $\phi 1$, and $\phi 2$.

$(1/f_{CLK} \leq T_{BUSY} \leq 3/f_{CLK})$

- **Detailed Explanation**

(1) Display ON/OFF

	R/W	D/I	DB7							DB0
Code	0	0	0	0	1	1	1	1	1	D

← high-order-bit low-order-bit →

The display data appears when D is 1 and disappears when D is 0. Though the data is not on the screen with D=0, it remains in the display data RAM. Therefore, you can make it appear by changing D=0 into D=1.

(2) Display start line

	R/W	D/I	DB7							DB0
Code	0	0	1	1	A	A	A	A	A	A

← high-order-bit low-order-bit →

Z address AAAAAA (binary) of the display data RAM is set at the display start line register and displayed at the top of the screen.
Fig. 7 are the examples of display (1/64 duty) when the start line=0 ∿ 3.
When the display duty is 1/64 or more (ex. 1/32, 1/24 etc.), the data of total line number of LCD screen, from the line specified by display start line instruction, is displayed.

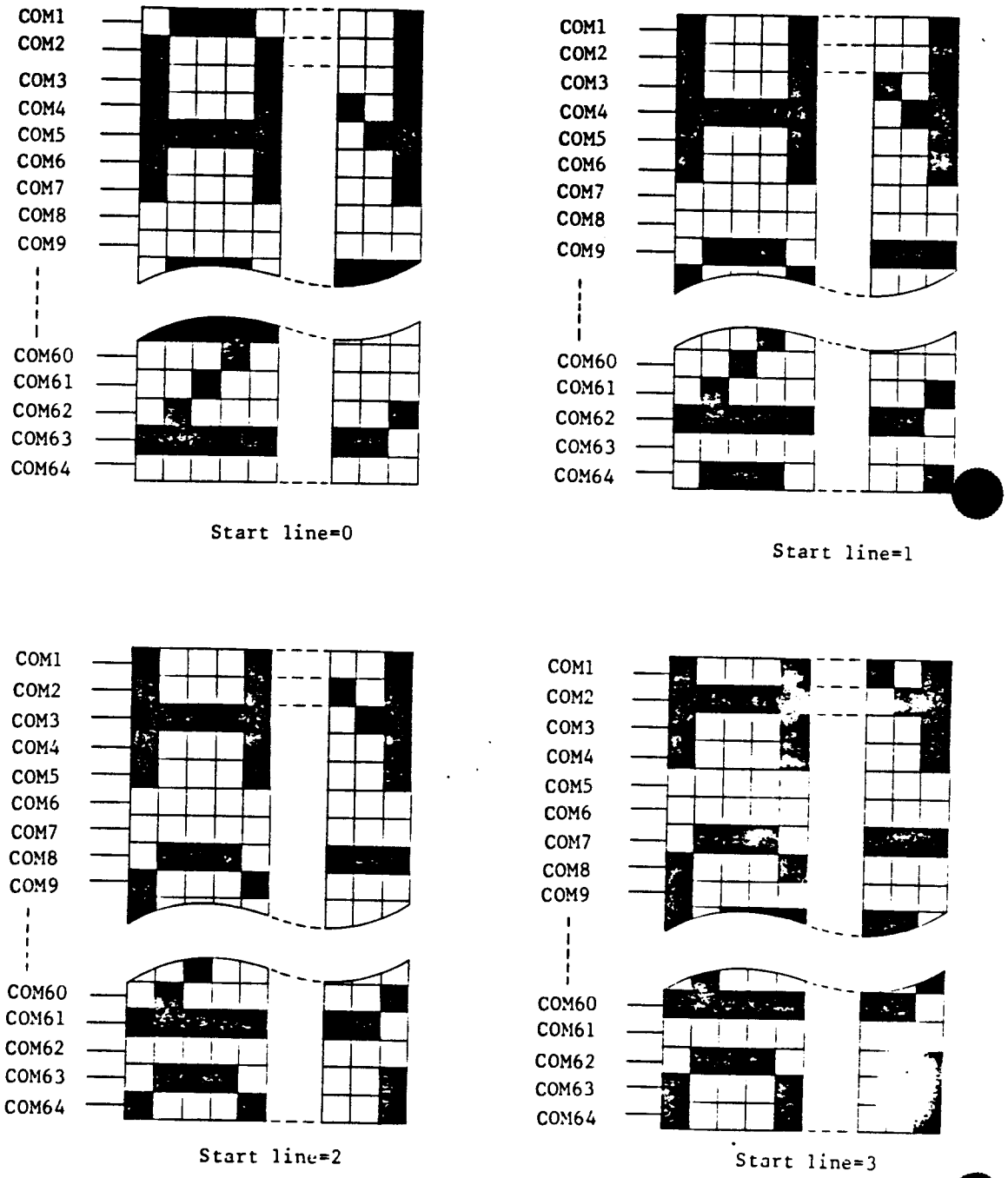


Fig. 7 Relation Between Start Line and Display

(3) Set page (X address)

	R/W	D/I	DB7							DB0
Code	0	0	1	0	1	1	1	A	A	A

← high-order-bit low-order-bit →

X address AAA (binary) of the display data RAM is set at the X address register. After that, writing or reading to or from MPU is executed in this specified page until the next page is set.

(4) Set Y address

	R/W	D/I	DB7							DB0
Code	0	0	0	1	A	A	A	A	A	A

← high-order-bit low-order-bit →

Y address AAAAAA (binary) of the display data RAM is set at the Y address counter. After that, Y address counter is increased by 1 every time the data is written or read to or from MPU.

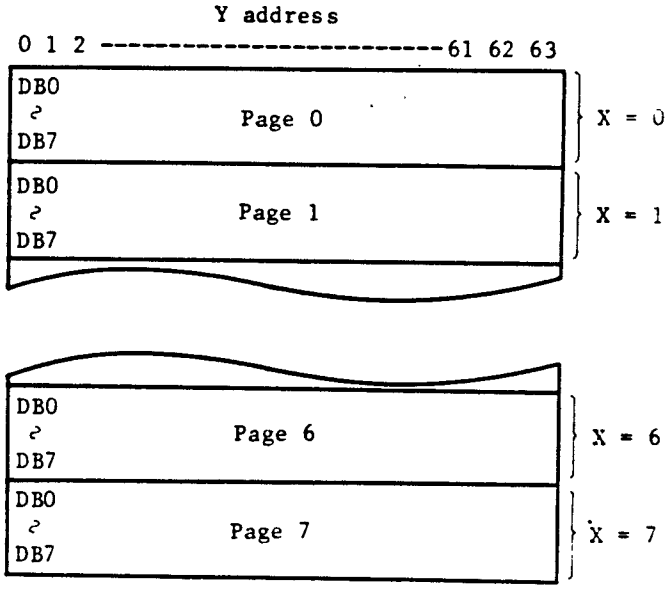


Fig. 8 Address Configuration of Display Data RAM

(5) Status Read

	R/W	D/I	DB7							DB0
Code	1	0	BUSY	0	ON/ OFF	RESET	0	0	0	0

← high-order-bit low-order-bit →

BUSY: When BUSY is 1, the LSI is in internal operation. No instructions are accepted while BUSY is 1, so you should make sure that BUSY is 0 before writing the next instruction.

ON/OFF: This bit shows the liquid crystal display conditions - ON condition or OFF condition.
When ON/OFF is 1, the display is in OFF condition.
When ON/OFF is 0, the display is in ON condition.

RESET: RESET=1 shows that the system is being initialized. In this condition, any instructions except Status Read instruction cannot be accepted.
RESET=0 shows that initializing has finished and the system is in the usual operation.

(6) Write Display Data

	R/W	D/I	DB7							DB0
Code	0	1	D	D	D	D	D	D	D	D

← high-order-bit low-order-bit →

Writes 8-bit data DDDDDDDD (binary) into the display data RAM. Then Y address is increased by 1 automatically.

(7) Read Display Data

	R/W	D/I	DB7							DB0
Code	1	1	D	D	D	D	D	D	D	D

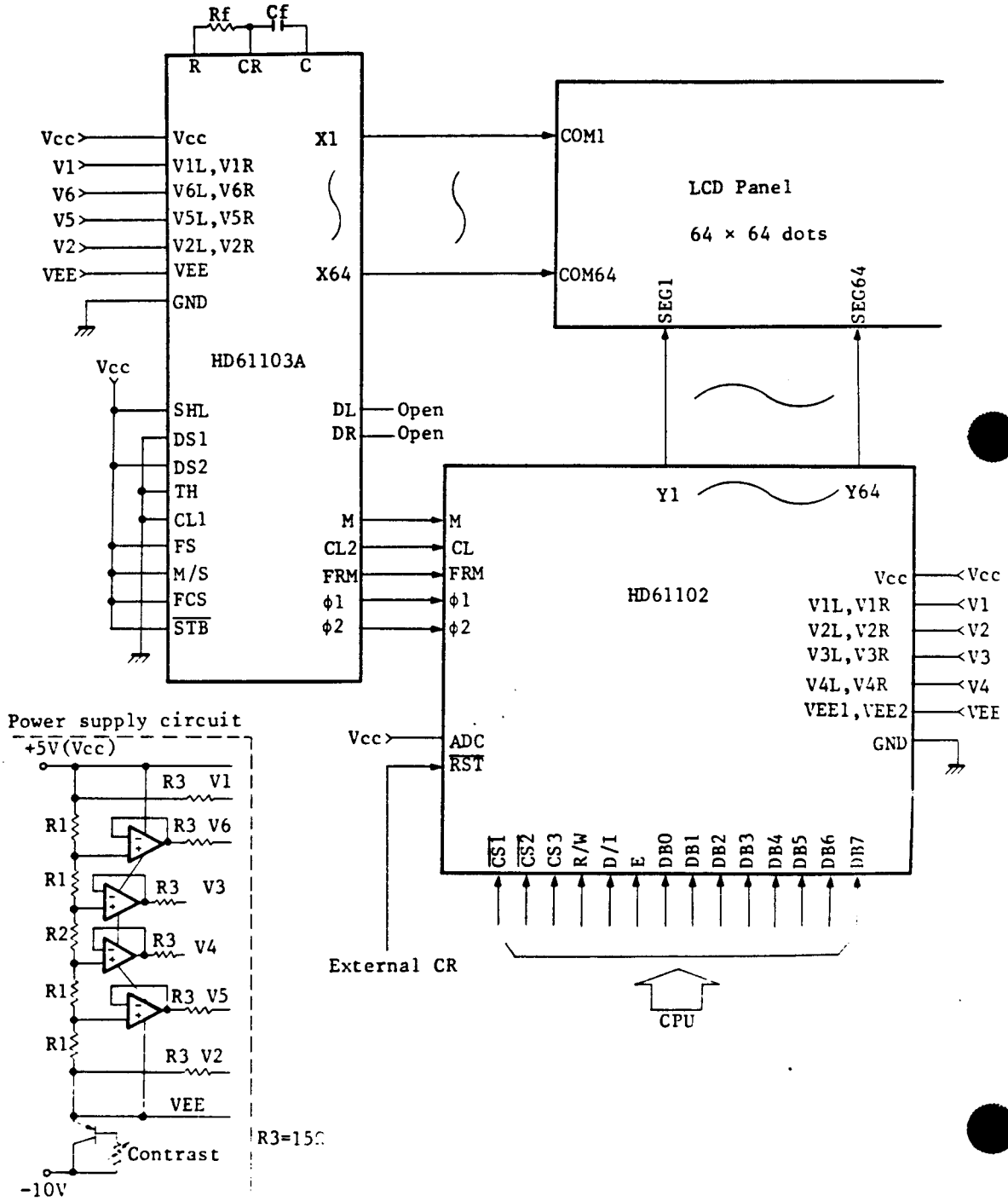
← high-order-bit low-order-bit →

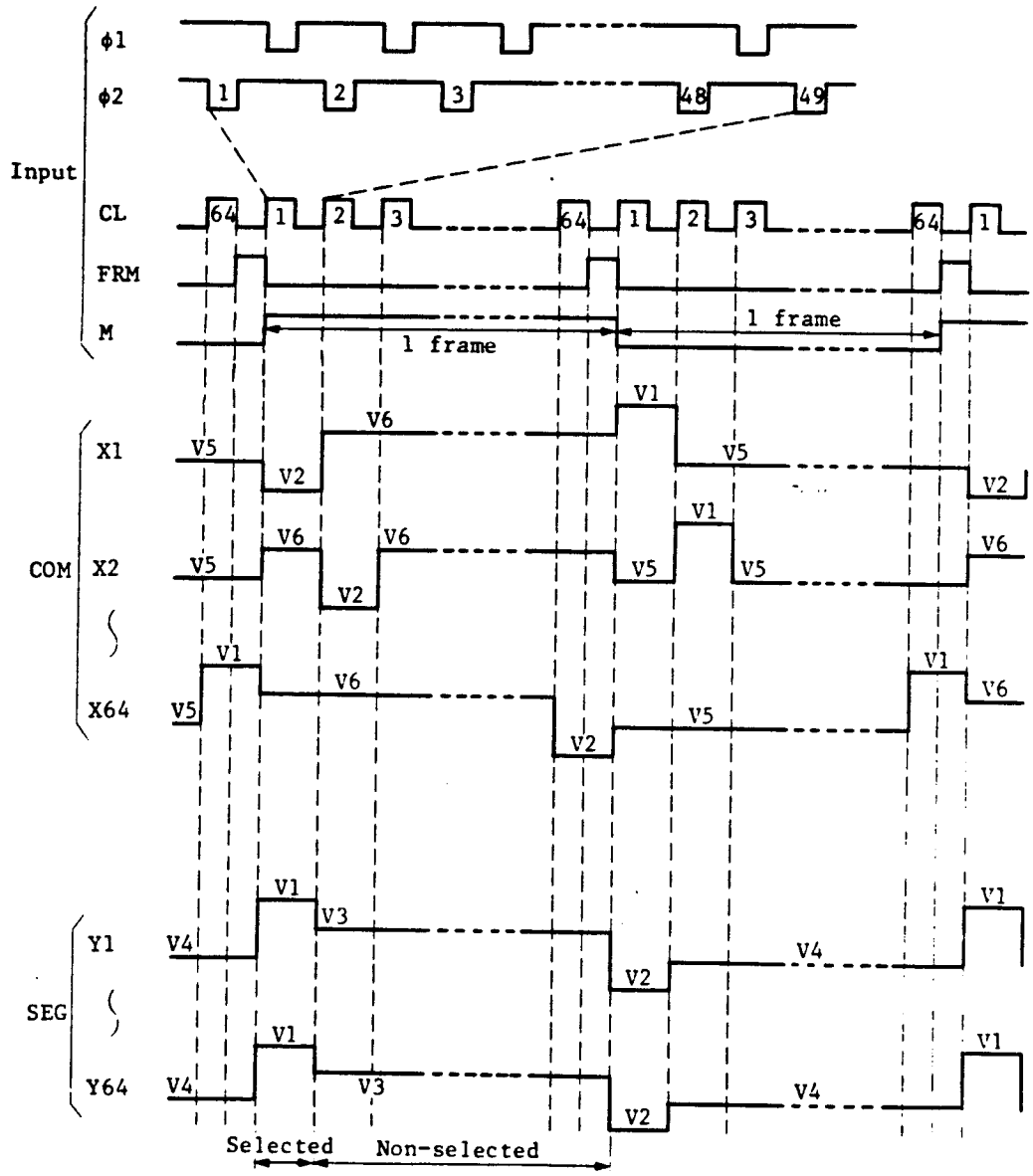
Read out 8-bit data DDDDDDDD (binary) from the display data RAM. Then Y address is increased by 1 automatically.

One dummy read is necessary soon after the address setting. For details, refer to the explanation of output register in "FUNCTION OF EACH BLOCK".

THE USAGE OF HD61102

- Interface with HD61103A (1/64 duty)



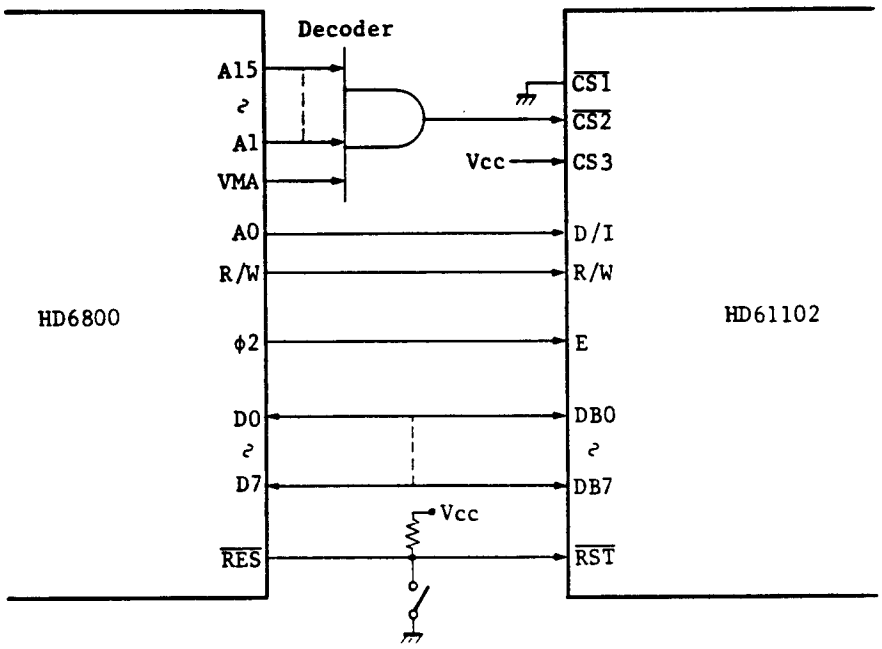


The wave forms of Y1 to Y64 outputs vary with the display data. In this example, the top line of the panel lights up and other dots do not.

Fig. 9 LCD Driver Timing Chart (1/64 duty)

● Interface with CPU

a) Example of connection with HD6800



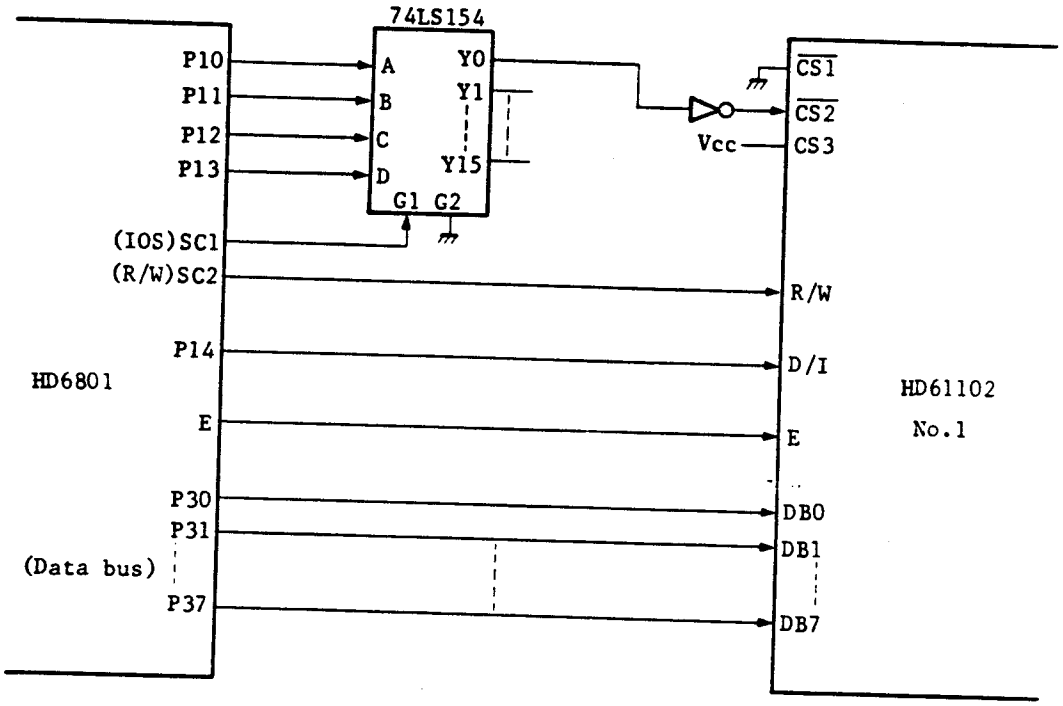
The example of connection with HD6800 series

In this decoder, addresses of HD61102 in the address area of HD6800 are:

Read/Write of the display data	$FFFF
Write of display instruction	$FFFE
Read out of status	$FFFE

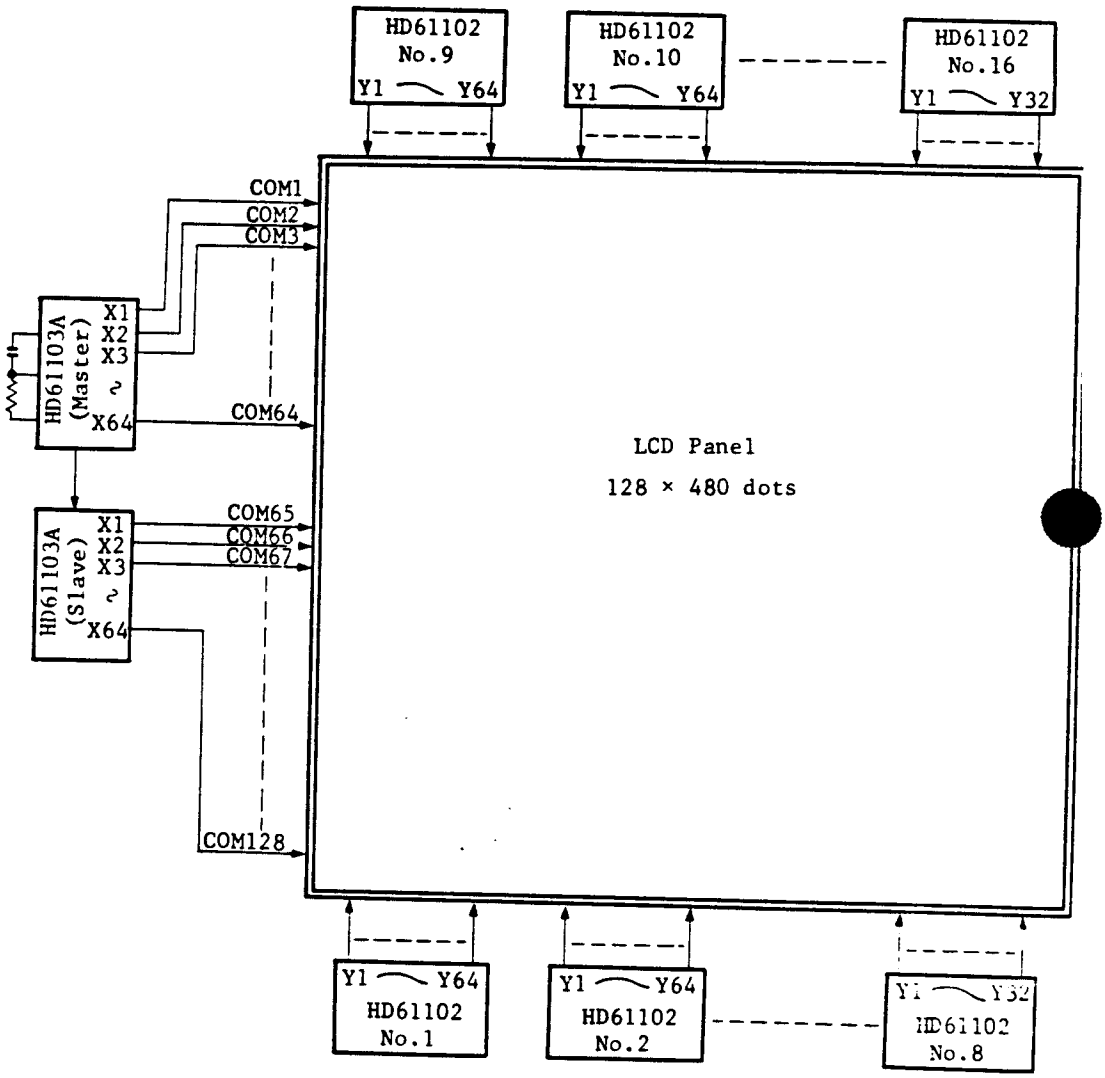
Therefore, you can control HD61102 by reading/writing the data at these addresses.

b) Example of connection with HD6801



- Set HD6801 in Mode 5. P10 to P14 are used as the output port and P30 to P37 as the data bus.
- 74LS154 is 4 to 16 decoder and genelate chip select signal to make specified HD61102 active after decoding 4 bits of P10 to P13.
- Therefore, after making the operation possible by P10 to P13 and specifying D/I signal by P14, read/write from/to the external memory area ($0100 to $01FE) to control HD61102. In this case, IOS signal is output from SC1 and R/W signal from SC2.
- For details of HD6800 and HD6801, refer to the each manual.

- Example of Application



Note) In this example, two HD61103A's output the equivalent waveforms. So, stand-alone operation is possible. In this case, connect COM1 COM65 to X1, COM2 and COM66 to X2, ..., and COM64 and COM128 to X6 However, for the large screen display, you had better drive in 2 row as this example to guarantee the display quality.

Appendixes

A

Resident Debugger

The ROM-resident debugger provides a way to debug application programs written in assembly language. It offers a way to examine or modify data in memory and the CPU registers, set a breakpoint, or single-step an assembly-language program.

The debugger is entered by typing RDB76 [ENTER] at the command mode prompt. The debugger must not use interrupts, so keys are more difficult to enter than in command mode. There is no cursor because cursor blink requires interrupts. Note that the debugger will *not* time out and turn the HP-94 off.

Table A-1. Resident Debugger Commands

Command	Description	Page
D	Display the contents of memory in hexadecimal characters.	A-4
G	Execute code until a breakpoint.	A-6
I *	Input data from an I/O port.	A-7
L †	Enter data in MDS format.	A-8
M	Display or change the contents of memory.	A-9
O *	Output data to an I/O port.	A-10
R	Display or change the contents of CPU registers.	A-11
S	Single-step execution of a program.	A-12
X	Switch the debugger console between HP-94 keyboard and serial port.	A-13

* For the I and O commands, press the [K] or [L] key respectively on the HP-94 keyboard. The command letter is shown in the display.

† The L command can only be entered if the console is set to the serial port.

All the characters recognized by the debugger can be entered without using [SHIFT], including the digit keys [0] through [9]. Some characters are assigned to different keys because the key which has that character printed on it is also a digit key.

Table A-2. Resident Debugger Keyboard Map

Key	Response
SPACE	:
.	,
#	+
K	I
L	O
Q	P
SHIFT	(ignored)

The ROM-resident debugger uses the HP-94 keyboard and display as the console. If the serial port is not used by the code being debugged, the serial port can be used as a console by connecting a terminal to the port. The port configuration is unconditionally set to 9600 baud, 7-bit data with even parity, and one stop bit.

Command Syntax

A parameter enclosed by [ ] is optional and may be omitted.

Underlined characters are characters displayed by the debugger.

In this chapter, the term word means a 16-bit value.

Example:

To input 40:1F, press these keys:

[4] [0] [SPACE] [1] [F]

An address has the following format:

[SSSS:]FFFF

SSSS is the segment expression (default = CS register)
FFFF is the offset expression

Both segment and offset can combine hexadecimal constants or two-character register names in addition or subtraction expressions using the + and - operators.

If more than four hexadecimal digits are entered, only the last four digits are used.

Valid register names are AX, BX, CX, DX, SP, BP, SI, DI, CS, DS, SS, ES, IP, and FL (flags).

Examples:

`41-1:145+AF` is interpreted as `40:1F4`.

`145-34+1` is interpreted as `CS:112`.

`IP+2` (if `IP` is 110h) is interpreted as `CS:112`.

`FFF0145-34+1` is interpreted as `CS:112` (`0145-34+1`).

`ES:BX+1` is interpreted as expected.

D

Display the contents of memory.

Syntax:

D[W]*address1*[, *address2*]ENTER

Description:

The D command displays in hexadecimal the contents of memory from *address1* to *address2*. If the W option is specified, the display is grouped by words; otherwise the display is grouped by bytes.

Console is the HP-94 keyboard/display:

The contents of memory up to a paragraph boundary (xxxx:xxx0) are displayed. The debugger then waits for a key to be pressed. If ENTER is pressed, the contents of the next 16 bytes of memory are displayed. Pressing any other key terminates the D command.

Console is the serial port:

The contents of memory are displayed in hexadecimal. If the W option is not specified, the corresponding ASCII characters are also displayed. Pressing any key terminates the D command. That key is then processed as a debugger command. Since software handshake characters are interpreted as keys, a handshake character such as XOFF from a terminal will terminate the D command.

Example 1: Console is HP-94 keyboard/display; display 0 : 0 through 0 : 14 as bytes.

```
*D0:0,14 (D 0 SPACE 0 . 1 4 ENTER)
0000:0000
 5B 0C 00 FC E1 0B
 00 FC 5B 0C 00 FC
 5B 0C 00 FC ENTER
0000:0010
 88 FF 95 7F 07
*
```

Example 2: Console is serial port; display 0 : 0 through 0 : 14 as bytes.

```
*D0:0,14
0000:0000  5B 0C 00 FC E1 0B 00 FC 5B 0C 00 FC 5B 0C 00 FC    [.......[...[...
0000:0010  88 FF 95 7F 07                                     .....
*
```

Example 3: Console is HP-94 keyboard/display; display DS : 0 through DS : 14 as words. Assume DS is 0.

```
*DWDS:0,14 (D W D S SPACE 0 . 1 4 ENTER)
0000:0000
 0C5B FC00 0BE1 FC00
 0C5B FC00 0C5B FC00 ENTER
0000:0010
 FF88 7F95 FF07
*
```

Example 4: Console is serial port; display DS : 0 through DS : 14 as words. Assume DS is 0.

```
*DW DS:0,14
0000:0000  0C5B FC00 0BE1 FC00 0C5B FC00 0C5B FC00
0000:0010  FF88 7F95 FF07
*
```

G

Execute code until a breakpoint.

Syntax:

G cccc:iiii- dd [*start address*][, *break address*] ENTER

Description:

The G command displays in hexadecimal the contents of CS:IP and the contents of the byte at that location. If a *break address* is specified, a breakpoint is set by writing an INT 3 (CCh) at that address. If the INT 3 cannot be written, an error occurs. This means it is not possible to set a breakpoint in a program in ROM. If a *start address* is not specified, program execution starts at CS:IP. If a *start address* is specified, program execution starts at that address.

When a program reaches the breakpoint, the debugger displays the following message and waits for another command.

```
BR@cccc:iiii
```

cccc is the value of the current CS register.
iiii is the value of the current IP register.

Note that if a program never reaches the breakpoint, the INT 3 remains in the code. The debugger will try to restore the instruction replaced by the INT 3 when the debugger is reentered. Because the program being debugged may have moved, it is recommended that a breakpoint not be set before returning to the operating system from the debugger.

Input data from a port.

Syntax:

`I` [W]*port*,

Description:

The `I` command inputs data from port *port* and displays it in hexadecimal. If the W option is specified, one word of data is input from *port* and displayed; otherwise one byte of data is input from *port* and displayed. Data is input from *port* and displayed each time a comma is entered.

Pressing [ENTER] terminates the `I` command.

L

Enter data in MDS format.

Syntax:

L[*bias*]ENTER

Description:

The L command inputs data in MDS format and loads the data to main memory. The data is written at the record address contained in the MDS data records added to the value of *bias*. The default *bias* is zero. The segment value can be set with a type 2 MDS record. The default segment is zero.

The L command is available only if the console is the serial port.

The L command discards any data received until the first colon (:) of the MDS file is encountered. If the data which follows the colon is not in MDS format, the L command terminates.

Display or change the contents of memory.

Syntax:

M[W]*address*, dd- [*new data*],

Description:

The **M** command displays in hexadecimal the contents of memory at *address*. If the W option is specified, memory is processed in words; otherwise memory is processed in bytes.

When a comma is entered, the contents of the next memory location are displayed.

Pressing [ENTER] terminates the **M** command.

If *new data* is specified (in hexadecimal), it is written to the memory location currently displayed. A read-after-write check is done to ensure that the data was written correctly. If the data read back does not match the data which was written, such as when trying to write to ROM, the **M** command terminates.

O

Output data to a port.

Syntax:

O[W]*port*, *data* [ENTER]
or
O[W]*port*, *data* ,

Description:

The O command outputs *data* to the specified *port*. If the W option is specified, one word of data is output to *port*; otherwise one byte of data is output to *port*.

If the O command is entered with a trailing comma, it writes the data to the port, then prompts for new data with a dash (–).

Pressing [ENTER] terminates the O command.

Display or change the contents of CPU registers.

Syntax:

R [ENTER]
or
R*register*-dddd- [*data*][,][ENTER]

Description:

The R command displays the contents of CPU registers in hexadecimal.

If *register* is specified, the contents of that register are displayed in hexadecimal. If *data* is specified (in hexadecimal), the register is changed to that value. A comma (,) continues on to the next register, if any; [ENTER] terminates the R command.

If *register* is not specified, the contents of all the CPU registers are displayed. The format depends on whether the console is the HP-94 or the serial port:

Console is the HP-94 keyboard/display:

```
*R [ENTER]
 AX=024A BX=0000
 CX=1FA2 DX=000E
 SP=07F2 BP=0250
 SI=0410 DI=0015 [ENTER]

 CS=0128 DS=0128
 SS=1F80 ES=1F80
 IP=0008 FL=F206
*
```

Console is the serial port:

```
*R [ENTER]
 AX=024A BX=0000 CX=1FA2 DX=000E SP=07F2 BP=0250 SI=0410
 DI=0015 CS=0128 DS=0128 SS=1F80 ES=1F80 IP=0008 FL=F206
*
```

S

Single-step execution of a program.

Syntax:

`S cccc:iiii- dd` [*start address*],

Description:

The `S` command displays the current `CS:IP` in hexadecimal and waits for another key. If *start address* is specified, the current `CS:IP` is set to that address.

A single instruction at the current `CS:IP` is executed when a comma (`,`) is entered. The `S` command displays the new `CS:IP` and waits for another key.

Single-step execution terminates when the [ENTER] key is pressed.

NOTE Because the HP-94 has a timer which interrupts every 5 ms, there will almost always be a pending interrupt when single-stepping code. Because all registers are restored before execution, including `FL`, interrupts are enabled unless the `FL` register has been modified to disable interrupts. When using the HP-94 keyboard, there is no key sequence to directly type the letter L. To view the `FL` register using the R command, type `RIP`, ([R][K][Q][.]).

Switch the debugger console between the HP-94 keyboard and serial port.

Syntax:

X [ENTER]

Description:

The X command switches the debugger console between the HP-94 keyboard/display and the serial port. Several commands display information in a format which is easier to read when the console is the serial port.

The X command displays the verification prompt "Ok ? " and waits for a key. If [Y] [ENTER] is entered, the console is switched to the serial port if the console was the HP-94 keyboard/display, or to the HP-94 keyboard/display if the console was the serial port.

When the console is switched to the serial port, the port is set to 9600 baud, even parity, 7-bit data, and one stop bit. The debugger operates without any hardware or software handshaking. Any handshaking characters sent by the terminal will be interpreted as keys, and will have the same effect as pressing keys. This is especially important for the D command.

The console must not be switched to the serial port while an application program which uses the serial port is being debugged.

CAUTION If the console is switched to the serial port which is connected to a terminal that cannot communicate at 9600 baud, even parity, 7-bit data, and one stop bit, or if the console is switched with no terminal attached, the only way to regain control of the HP-94 is to press the reset switch.

B

Errors

Table B-1. Operating System Errors

Hex	Decimal	Meaning
64h	100 *	BASIC interpreter not found
65h	101	Illegal parameter
66h	102	Directory does not exist
67h	103	File not found
68h	104	Too many files
69h	105	Channel not open
6Ah	106	Channel already open
6Bh	107	File already open
6Ch	108	File already exists
6Dh	109	Read-only access
6Eh	110	Access restricted
6Fh	111	No room for file
70h	112	No room to expand file
71h	113	No room for scratch area
72h	114	Scratch area does not exist
73h	115 †	Short record detected
74h	116 †	Terminate character detected
75h	117 †	End-of-data
76h	118	Timeout
77h	119	Power switch pressed
C8h	200	Low battery
C9h	201	Receive buffer overflow
CAh	202	Parity error
CBh	203	Overrun error
CCh	204	Parity and overrun error
CDh	205	Framing error
CEh	206	Framing and parity error
CFh	207	Framing and overrun error
D0h	208	Framing, overrun, and parity error
D1h	209 †	Invalid MDS file received
D2h	210 *	Low backup battery — main memory
D3h	211 *	Low backup battery — 128K memory board or 40K RAM card
D4h	212 *	Checksum error — main memory directory table
D5h	213 *	Checksum error — 40K RAM or ROM/EPROM card directory table
D6h	214 *	Checksum error — reserved scratch space
D7h	215 *	Checksum error — main memory free space
D8h	216 *	Checksum error — main memory file
D9h	217 *	Checksum error — 40K RAM or ROM/EPROM card file
DAh	218	Lost connection while transmitting
DBh	219 †	Illegal use of operating system stack

* Only reported when machine is turned on.

† Never reported by built-in BASIC keywords.

Table B-2. BASIC Interpreter Errors

Message	Meaning
AR	Array subscript error
BM	BASIC interpreter malfunction
BR	Branch destination error
CN	Data conversion error
CO	Conversion overflow
DO	Decimal overflow
DT	Data error
EP	Missing END statement
FN	Illegal DEF FN statement
IL	Illegal argument
IR	Insufficient RAM
IS	Illegal statement
LN	Nonexistent line
MO	Memory overflow
NF	Program not found
RT	RETURN or SYRT error
SY	Syntax error
TY	Data type mismatch
UM	Unmatched number of arguments

C

Keyboard Layout

Table C-1. ASCII Characters and Keycodes for Each Key

Shifted Key (orange)	Shifted Character	Unshifted Key (white)	Unshifted Character	Keycode
A	A (41h)	(unmarked)	user-defined (80h)	01h
B	B (42h)	(unmarked)	user-defined (81h)	06h
C	C (43h)	(unmarked)	user-defined (82h)	0Bh
D	D (44h)	(unmarked)	user-defined (83h)	10h
E	E (45h)	(unmarked)	user-defined (84h)	02h
F	F (46h)	(unmarked)	user-defined (85h)	07h
G	G (47h)	(unmarked)	user-defined (86h)	0Ch
H	H (48h)	7	7 (37h)	11h
I	I (49h)	8	8 (38h)	16h
J	J (4Ah)	9	9 (39h)	1Bh
K	K (4Bh)	(unmarked)	user-defined (87h)	03h
L	L (4Ch)	(unmarked)	user-defined (88h)	08h
M	M (4Dh)	(unmarked)	user-defined (89h)	0Dh
N	N (4Eh)	4	4 (34h)	12h
O	O (4Fh)	5	5 (35h)	17h
P	P (50h)	6	6 (36h)	1Ch
Q	Q (51h)	(unmarked)	user-defined (8Ah)	04h
R	R (52h)	(unmarked)	user-defined (8Bh)	09h
S	S (53h)	(unmarked)	user-defined (8Ch)	0Eh
T	T (54h)	1	1 (31h)	13h
U	U (55h)	2	2 (32h)	18h
V	V (56h)	3	3 (33h)	1Dh
W	W (57h)	(unmarked)	user-defined (8Dh)	05h
X	X (58h)	(unmarked)	user-defined (8Eh)	0Ah
Y	Y (59h)	(unmarked)	user-defined (8Fh)	0Fh
Z	Z (5Ah)	0	0 (30h)	14h
*	* (2Ah)	#	# (23h)	15h
SPACE	(space) (20h)	00	00 (30h 30h)	19h
—	— (2Dh)	—	— (2Dh)	1Ah
.	. (2Eh)	.	. (2Eh)	1Eh
SHIFT	(none)	SHIFT	(none)	1Fh
CLEAR	(CAN) (18h)	CLEAR	(CAN) (18h)	20h
←	(DEL) (7Fh)	←	(DEL) (7Fh)	21h
ENTER	(CR) (0Dh)	ENTER	(CR) (0Dh)	22h

D

Roman-8 Character Set

ASCII Char.	Character Code			
	Hex	Dec	Oct	Binary
NUL	00	0	000	00000000
SOH	01	1	001	00000001
STX	02	2	002	00000010
ETX	03	3	003	00000011
EOT	04	4	004	00000100
ENQ	05	5	005	00000101
ACK	06	6	006	00000110
BEL	07	7	007	00000111
BS	08	8	010	00001000
HT	09	9	011	00001001
LF	0A	10	012	00001010
VT	0B	11	013	00001011
FF	0C	12	014	00001100
CR	0D	13	015	00001101
SO	0E	14	016	00001110
SI	0F	15	017	00001111
DLE	10	16	020	00010000
DC1	11	17	021	00010001
DC2	12	18	022	00010010
DC3	13	19	023	00010011
DC4	14	20	024	00010100
NAK	15	21	025	00010101
SYN	16	22	026	00010110
ETB	17	23	027	00010111
CAN	18	24	030	00011000
EM	19	25	031	00011001
SUB	1A	26	032	00011010
ESC	1B	27	033	00011011
FS	1C	28	034	00011100
GS	1D	29	035	00011101
RS	1E	30	036	00011110
US	1F	31	037	00011111

ASCII Char.	Character Code			
	Hex	Dec	Oct	Binary
space	20	32	040	00100000
!	21	33	041	00100001
"	22	34	042	00100010
#	23	35	043	00100011
$	24	36	044	00100100
%	25	37	045	00100101
&	26	38	046	00100110
'	27	39	047	00100111
(	28	40	050	00101000
)	29	41	051	00101001
*	2A	42	052	00101010
+	2B	43	053	00101011
,	2C	44	054	00101100
-	2D	45	055	00101101
.	2E	46	056	00101110
/	2F	47	057	00101111
0	30	48	060	00110000
1	31	49	061	00110001
2	32	50	062	00110010
3	33	51	063	00110011
4	34	52	064	00110100
5	35	53	065	00110101
6	36	54	066	00110110
7	37	55	067	00110111
8	38	56	070	00111000
9	39	57	071	00111001
:	3A	58	072	00111010
;	3B	59	073	00111011
<	3C	60	074	00111100
=	3D	61	075	00111101
>	3E	62	076	00111110
?	3F	63	077	00111111

ASCII Char.	Character Code			
	Hex	Dec	Oct	Binary
@	40	64	100	01000000
A	41	65	101	01000001
B	42	66	102	01000010
C	43	67	103	01000011
D	44	68	104	01000100
E	45	69	105	01000101
F	46	70	106	01000110
G	47	71	107	01000111
H	48	72	110	01001000
I	49	73	111	01001001
J	4A	74	112	01001010
K	4B	75	113	01001011
L	4C	76	114	01001100
M	4D	77	115	01001101
N	4E	78	116	01001110
O	4F	79	117	01001111
P	50	80	120	01010000
Q	51	81	121	01010001
R	52	82	122	01010010
S	53	83	123	01010011
T	54	84	124	01010100
U	55	85	125	01010101
V	56	86	126	01010110
W	57	87	127	01010111
X	58	88	130	01011000
Y	59	89	131	01011001
Z	5A	90	132	01011010
[	5B	91	133	01011011
\	5C	92	134	01011100
]	5D	93	135	01011101
^	5E	94	136	01011110
_	5F	95	137	01011111

ASCII Char.	Character Code			
	Hex	Dec	Oct	Binary
`	60	96	140	01100000
a	61	97	141	01100001
b	62	98	142	01100010
c	63	99	143	01100011
d	64	100	144	01100100
e	65	101	145	01100101
f	66	102	146	01100110
g	67	103	147	01100111
h	68	104	150	01101000
i	69	105	151	01101001
j	6A	106	152	01101010
k	6B	107	153	01101011
l	6C	108	154	01101100
m	6D	109	155	01101101
n	6E	110	156	01101110
o	6F	111	157	01101111
p	70	112	160	01110000
q	71	113	161	01110001
r	72	114	162	01110010
s	73	115	163	01110011
t	74	116	164	01110100
u	75	117	165	01110101
v	76	118	166	01110110
w	77	119	167	01110111
x	78	120	170	01111000
y	79	121	171	01111001
z	7A	122	172	01111010
{	7B	123	173	01111011
|	7C	124	174	01111100
}	7D	125	175	01111101
~	7E	126	176	01111110
DEL	7F	127	177	01111111

ASCII Char.	Character Code			
	Hex	Dec	Oct	Binary
	80	128	200	10000000
	81	129	201	10000001
	82	130	202	10000010
	83	131	203	10000011
	84	132	204	10000100
	85	133	205	10000101
	86	134	206	10000110
	87	135	207	10000111
	88	136	210	10001000
	89	137	211	10001001
	8A	138	212	10001010
	8B	139	213	10001011
	8C	140	214	10001100
	8D	141	215	10001101
	8E	142	216	10001110
	8F	143	217	10001111
	90	144	220	10010000
	91	145	221	10010001
	92	146	222	10010010
	93	147	223	10010011
	94	148	224	10010100
	95	149	225	10010101
	96	150	226	10010110
	97	151	227	10010111
	98	152	230	10011000
	99	153	231	10011001
	9A	154	232	10011010
	9B	155	233	10011011
	9C	156	234	10011100
	9D	157	235	10011101
	9E	158	236	10011110
	9F	159	237	10011111

ASCII Char.	Character Code			
	Hex	Dec	Oct	Binary
space	A0	160	240	10100000
À	A1	161	241	10100001
Â	A2	162	242	10100010
È	A3	163	243	10100011
Ê	A4	164	244	10100100
Ë	A5	165	245	10100101
Î	A6	166	246	10100110
Ï	A7	167	247	10100111
´	A8	168	250	10101000
`	A9	169	251	10101001
^	AA	170	252	10101010
¨	AB	171	253	10101011
~	AC	172	254	10101100
Ù	AD	173	255	10101101
Û	AE	174	256	10101110
£	AF	175	257	10101111
¯	B0	176	260	10110000
Ý	B1	177	261	10110001
ý	B2	178	262	10110010
°	B3	179	263	10110011
Ç	B4	180	264	10110100
ç	B5	181	265	10110101
Ñ	B6	182	266	10110110
ñ	B7	183	267	10110111
¡	B8	184	270	10111000
¿	B9	185	271	10111001
¤	BA	186	272	10111010
£	BB	187	273	10111011
¥	BC	188	274	10111100
§	BD	189	275	10111101
ƒ	BE	190	276	10111110
¢	BF	191	277	10111111

ASCII Char.	Character Code			
	Hex	Dec	Oct	Binary
â	C0	192	300	11000000
ê	C1	193	301	11000001
ô	C2	194	302	11000010
û	C3	195	303	11000011
á	C4	196	304	11000100
é	C5	197	305	11000101
ó	C6	198	306	11000110
ú	C7	199	307	11000111
à	C8	200	310	11001000
è	C9	201	311	11001001
ò	CA	202	312	11001010
ù	CB	203	313	11001011
ä	CC	204	314	11001100
ë	CD	205	315	11001101
ö	CE	206	316	11001110
ü	CF	207	317	11001111
Å	D0	208	320	11010000
î	D1	209	321	11010001
Ø	D2	210	322	11010010
Æ	D3	211	323	11010011
å	D4	212	324	11010100
í	D5	213	325	11010101
ø	D6	214	326	11010110
æ	D7	215	327	11010111
Ä	D8	216	330	11011000
ì	D9	217	331	11011001
Ö	DA	218	332	11011010
Ü	DB	219	333	11011011
É	DC	220	334	11011100
ï	DD	221	335	11011101
ß	DE	222	336	11011110
Ô	DF	223	337	11011111

ASCII Char.	Character Code			
	Hex	Dec	Oct	Binary
Á	E0	224	340	11100000
Ã	E1	225	341	11100001
ã	E2	226	342	11100010
Ð	E3	227	343	11100011
ð	E4	228	344	11100100
Í	E5	229	345	11100101
Ì	E6	230	346	11100110
Ó	E7	231	347	11100111
Ò	E8	232	350	11101000
Õ	E9	233	351	11101001
õ	EA	234	352	11101010
Š	EB	235	353	11101011
š	EC	236	354	11101100
Ú	ED	237	355	11101101
Ÿ	EE	238	356	11101110
ÿ	EF	239	357	11101111
Þ	F0	240	360	11110000
þ	F1	241	361	11110001
▪	F2	242	362	11110010
µ	F3	243	363	11110011
¶	F4	244	364	11110100
3/4	F5	245	365	11110101
–	F6	246	366	11110110
1/4	F7	247	367	11110111
1/2	F8	248	370	11111000
ª	F9	249	371	11111001
º	FA	250	372	11111010
<<	FB	251	373	11111011
■	FC	252	374	11111100
>>	FD	253	375	11111101
±	FE	254	376	11111110
▒	FF	255	377	11111111

E

Display Control Characters

Table E-1. Display Control Characters

Hex Value	Meaning
01h (SOH)	Turn on cursor.
02h (STX)	Turn off cursor.
06h (ACK)	High tone beep for 0.5 second.
07h (BEL)	Low tone beep for 0.5 second.
08h (BS)	Move cursor left one column. When the cursor reaches the left end of the line, it will back up to the right end of the previous line. When the cursor reaches the top left corner, backspace will have no effect.
0Ah (LF)	Move cursor down one line. If the cursor is on the bottom line, the display contents will scroll up one line.
0Bh (VT)	Clear every character from the cursor position to the end of the current line. The cursor position will be unchanged.
0Ch (FF)	Move cursor to upper left corner and clear the display.
0Dh (CR)	Move cursor to left end of current line.
0Eh (SO)	Change keyboard to numeric mode (underline cursor).
0Fh (SI)	Change keyboard to alpha mode (block cursor).
1Eh (RS)	Turn on display backlight.
1Fh (US)	Turn off display backlight.

F

Memory Map

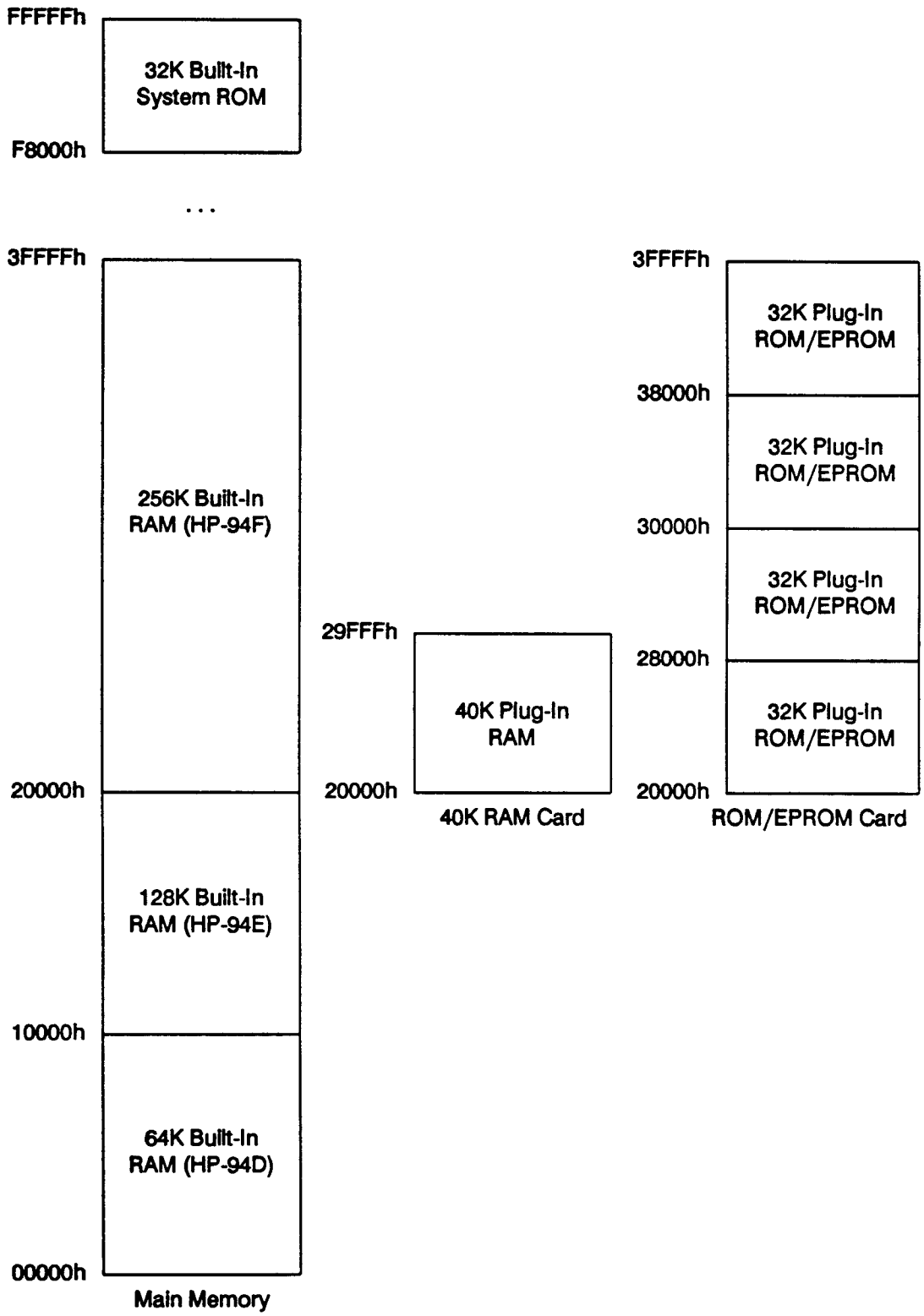


Figure F-1. Memory Map of the HP-94

G

Control and Status Register Addresses

Table G-1. I/O Addresses for Control and Status Registers

I/O Address	Register Name	Read/ Write
00h	Interrupt Control	W
00h	Interrupt Status	R
01h	Interrupt Clear	W
01h	End of Interrupt	R
02h	System Timer Data	R/W
03h	System Timer Control	W
04h	Bar Code Timer Data (lower 8 bits)	R/W
05h	Bar Code Timer Data (upper 4 bits)	R/W
06h	Bar Code Timer Control	W
07h	Bar Code Timer Value Capture	W
08h	Bar Code Timer Clear	W
0Ah	Baud Rate Clock Value	W
0Bh	Main Control	W
0Bh	Main Status	R
0Ch	Real-Time Clock Control	W
0Ch	Real-Time Clock Status/Data	R
0Eh	Keyboard Control	W
0Eh	Keyboard Status	R
10h	Serial Port Data	R/W
11h	Serial Port Control	W
11h	Serial Port Status	R
12h	Right LCD Driver Control	W
12h	Right LCD Driver Status	R
13h	Right LCD Driver Data	R/W
14h	Left LCD Driver Control	W
14h	Left LCD Driver Status	R
15h	Left LCD Driver Data	R/W
1Bh	Power Control	W

H

Hardware Interrupts

Table H-1. HP-94 Hardware Interrupts

Interrupt Type	Interrupt Name
50h	System Timer
51h	Bar Code Timer
52h	Bar Code Port Transition
53h	Serial Port Data Received
54h	Low Main Battery Voltage
55h	Power Switch
56h	Reserved Interrupt 1
57h	Reserved Interrupt 2

Highest

↓

Interrupt Priority

↓

Lowest

I

Operating System Functions

Table I-1. Operating System Function List

Name	Code	Description
BEEP	07h	Beep a high or low tone for specified duration
BUFFER_STATUS	06h	Get the number of bytes in or flush either the key buffer or the serial port handler receive buffer
CLOSE	10h	Close an I/O channel
CREATE	11h	Create a data file
CURSOR	05h	Read or change the cursor position on the LCD
DELETE	14h	Delete data file
DISPLAY_ERROR	18h	Display numeric error
END_PROGRAM	00h	Terminate the application program
FIND_FILE	16h	Find first occurrence of a file
FIND_NEXT	17h	Find subsequent occurrences of file
GET_CHAR	01h	Get a character from key buffer
GET_LINE	02h	Get a character string from the key buffer
GET_MEM	0Bh	Get a scratch area of memory
MEM_CONFIG	0Dh	Identify memory configuration
OPEN	0Fh	Open an I/O channel
PUT_CHAR	03h	Display a character on the LCD
PUT_LINE	04h	Display a character string on the LCD
READ	12h	Read data from an I/O channel
REL_MEM	0Ch	Release scratch area of memory
ROOM	0Eh	Identify available room in a directory
SEEK	15h	Move data file access pointer
SET_INTR	0Ah	Define power switch or low battery interrupt routines or disable/enable the power switch interrupt
TIMEOUT	09h	Set system or backlight timeout value
TIME_DATE	08h	Set or read the time and date on the real-time clock
WRITE	13h	Write data to an I/O channel

J

BASIC Interpreter Utility Routines

Table J-1. BASIC Interpreter Utility Routine List

Name	Offset	Description
ERROR	34h	Display error and end program
GETARG	3Ch	Convert real or integer into binary
IOERR	38h	Process errors in accordance with SYER
SADD	14h	Add two real numbers
SDIV	20h	Divide two real numbers
SETARG	40h	Convert binary into real or integer
SMUL	1Ch	Multiply two real numbers
SNEG	28h	Change sign of real number
SPOW	24h	Raise one real number to the power of another
SSUB	18h	Subtract two real numbers
TOBIN	30h	Convert integer or real into integer
TOREAL	2Ch	Convert integer or real into real

K

Program Resource Allocation

There are certain resources related to assembly language programs that must be chosen carefully to prevent conflict between different programs. Some of these resources are for any program, while others are for user-defined handlers only. These are as follows:

- Error Numbers
  These are used to report error conditions to calling BASIC or assembly language programs. BASIC programs can report numeric or non-numeric errors, although both internally map to an error number.
- Handler Identifier
  This is returned by the `IDENTIFY` function of the handler `IOCTL` routine.
- Valid Data Flag
  This is used to determine if the data in the parameter scratch area is correct for the handler being used.
- `IOCTL` Function Codes
  These are the function codes for the different functions in the handler `IOCTL` routine.

Refer to the "User-Defined Handlers" chapter in part 1, "Operating System", for details on the last three resources.

Below are tables summarizing usage of these resources by Hewlett-Packard programs. Remember that Hewlett-Packard also reserves `SY` as the first two characters of HP assembly language program and keyword names, and HN as the first two characters of HP handler names.

Table K-1. Error Number Usage

Error Number Range		Reserved For
Start	End	
00h (0)	00h (0)	No error
01h (1)	13h (19)	BASIC interpreter
64h (100)	77h (119)	Operating system
96h (150)	A9h (169)	*HP-94 Datacomm Utilities Pac*
C8h (200)	DBh (219)	Operating system

Table K-2. Hewlett-Packard Handler Resource Usage

Handler Name	Handler Identifier	Valid Data Flag	IOCTL Function Codes
HNBC	BC	FFh	00h-04h
HNSP	SP	FFh	00h-04h,80h
HNSG	SG	FEh	00h-04h,81h
Reserved	—	80h-FDh	05h-06h

To reserve resources for a particular program, a request should be made in writing to Hewlett-Packard. The request should indicate the resources required and their desired values. Also provide information about the software these resources will be used for (commercial applications for general sale, company-specific internal use only, etc.). This will help us allocate these limited resources as efficiently as possible. Send the request to:

Hewlett-Packard Portable Computer Division
Technical Marketing Software Support Group
1000 N.E. Circle Blvd.
Corvallis, OR 97330

If the desired value is available, it will be reserved for use by the program. If not, it will be necessary to select a different value for that resource.

L

Hewlett-Packard Bar Code Handlers

Hewlett-Packard supplies three bar code handlers with the *HP-94 Software Development System*:

- HNBC, a low-level bar code handler for the bar code port.
- HNSP, a low-level bar code handler for the serial port.
- HNWN, a high-level bar code handler for Hewlett-Packard Smart Wands (HP 39961D, HP 39963D, and HP 39965D).

These are all supplied as EXE files, and will all execute from RAM or ROM. This appendix will discuss details of these handlers important for assembly language programmers, including statistics, behavior of handler routines, errors, and parameter passing.

All three handlers follow the general behavior pattern described in the "User-Defined Handlers" chapter in part 1, "Operating System", so only the specific characteristics that are unique to each handler will be described here. This appendix assumes that the handler descriptions in the *HP-94 BASIC Reference Manual* have been read; that information will not be repeated here.

HNBC Low-Level Handler for Bar Code Port

HNBC is a low-level bar code handler for the bar code port. It is designed to allow "smart" bar code scanning devices to be connected to the bar code port — devices which do on-board decoding of bar code labels into ASCII, and return it as serial data. The HP-94 bar code port does not have a hardware UART to receive serial data, but HNBC performs the functions of a UART in software (assembling the serial bit stream into bytes, and checking for parity and framing errors).

HNBC is designed to work with bar code devices whose electrical characteristics match those of the HP-94 bar code port, and that send data in bursts of no more than 255 characters, with an intercharacter delay (time between characters) of 1-106 ms. HNBC is only supported for Hewlett-Packard Smart Wands (HP 39961D, HP 39963D, and HP 39965D).

HNBC Statistics

Here are pertinent statistics for HNBC.

Table L-1. HNBC Statistics

Item	Value
Version Number	1.00
Handler Identifier	BC
Valid Data Flag	FFh
Length in HP-94	2151 bytes
Scratch Areas Used	1 of 288 bytes *
Handler Information Table Offsets Used	00h, 04h
Valid Channel Numbers	2

* One additional 16-byte parameter scratch area is allocated if it was not allocated before opening the handler.

HNBC Capabilities

HNBC provides the following capabilities:

- Read-Only Operation
  Because the bar code port is read-only, no data can be written to it.
- Good Read Beep
  Automatic beep on successful decoding of a bar code label.
- Key Abort
  Allows a key being pressed to abort waiting for a successful scan.
- Received-Data Buffering
  Received data is placed in a 255-byte buffer. There is no transmit buffer.
- Speeds
  Speeds can be set from 150 to 9600 baud.
- Data Bits
  Seven only.
- Parity
  Zero, one, even, or odd parity.
- Stop Bits
  One only. For a receive-only port, all stop bits received after the first one are treated as intercharacter delay.
- Terminate Character Control
  When defined, a received terminate character will signal the end of the bar code data from the scanning device.

The table below describes how HNBC behaves. It shows the action taken by the handler routines as well as during its interrupt service routine, not including normal handler activities described in the "User-Defined Handlers" chapter. Note that certain actions, such as beeping on a good scan or responding to a received terminate character, will only occur if the appropriate options were enabled when the handler was opened.

Table L-2. Behavior of HNBC

Routine	Activities
CLOSE	Disable interrupt 52h and restore interrupt vector Turn off power to bar code port Release handler scratch area
IOCTL	Implement the reserved IOCTL functions 00h-04h
OPEN	Allocate parameter scratch area if needed Allocate handler scratch area Take over interrupt 52h vector Initialize operating configuration * Supply power to bar code port Return handler scratch area address in CX †
POWERON	Do nothing
READ	Wait for key up before accepting data Report error CDh (205) if no bar code device connected Discard data until none for 106 ms to avoid reading middle of label Return no data and error 75h (117) if read aborted by pressing key Monitor and report low battery, power switch, and timeout errors ‡ Enable interrupt 52h Wait for first byte received End read operation if no subsequent data received for 106 ms Compute parity of received data Report error 74h (116) if terminate character detected Report error 75h (117) if scanned length less than requested length Report errors detected in interrupt service routine Issue high beep if scan successful Return data from receive buffer
RSVD2	Do nothing
RSVD3	Do nothing
TERM	Flush receive buffer
WARM	Perform all OPEN routine activities except those related to scratch areas
WRITE	Return error 6Dh (109).
Interrupt Service	Read bar code status from main control register for all bits in each byte Monitor framing and receive buffer overflow errors Accumulate data into receive buffer

* Baud rate, parity, key abort, good read beep, and terminate character.

† Handlers are not required to return this, but HNBC does.

‡ System timeout only monitored until first byte received. After that, no data received for 106 ms signals the end of the label. Consequently, all characters must be received in a burst in which the intercharacter delay (time between characters) must be less than 106 ms.

CAUTION The HP-94 is unable to receive data through the serial port while the READ routine is executing. READ prevents this from happening by disabling the serial port data received interrupt. If both the serial and bar code ports must be open simultaneously, programs should halt serial port input before calling the HNBC READ routine (perhaps by sending an XOFF).

While the READ routine is executing, the background timer routine (interrupt 1Ch) must not clear the CPU interrupt flag (CLI), write the interrupt control register (00h) to enable any interrupts, or issue software interrupts (such as interrupt 1Ah for the operating system functions). Doing so may cause loss of bar code data, resulting in parity or framing errors.

The background timer accuracy will be degraded if the baud rate of the bar code device is less than 2400 baud or if the device sends its data with an intercharacter delay of less than 1 ms.

The errors reported by HNBC are shown in the following table.

Table L-3. Errors Reported by HNBC

Routine	Errors
CLOSE	6Eh
IOCTL	65h
OPEN	65h,67h,6Eh,71h
POWERON	None
READ	74h,75h,76h,77h,C8h,C9h,CAh,CDh,CEh
RSVD2	None
RSVD3	None
TERM	None
WARM	None
WRITE	6Dh
Interrupt Service *	C9h,CDh

* Detected by interrupt service routine, but reported by READ routine.

NOTE Two errors reported by the READ routine (74h, terminate character detected, and 75h, end of data) do not indicate error conditions, but signal the end of bar code data for the BASIC GET # and INPUT # statements. Assembly language programs using HNBC should handle these two errors differently than other errors from the READ routine.

If READ has not transferred all the data in its receive buffer when any read error occurs, it will flush the buffer. The next time READ is called, it will wait for a new bar code scan.

Parameters at OPEN Time

When HNBC is opened, it looks at offset 04h of the handler information table. If the value is zero, it allocates a one-paragraph parameter scratch area, places the default configuration in it, and places the scratch area address in the table. If the value is non-zero, it uses the value as the segment address of an existing parameter scratch area, and reads the configuration to use from that scratch area. The meanings of the parameters are shown below. In these figures, the offsets are from the start of the parameter scratch area. A copy of these parameters are pointed to by ES:DX in the GET_CONFIG and CHANGE_CONFIG reserved IOCTL functions (01h and 02h).

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

Figure L-1. HNBC Valid Data Flag — Parameter Byte 1 (Offset 00h)

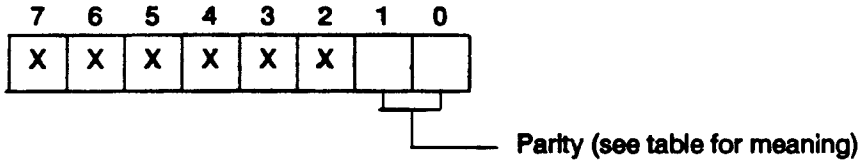
7	6	5	4	3	2	1	0
0	0	0	0	0			

Baud Rate (see table for meaning)

Figure L-2. HNBC Baud Rate — Parameter Byte 2 (Offset 01h)

Table L-4. HNBC Baud Rate Values

Value	Baud Rate
1	9600
2	4800
3	2400
4	1200
5	600
6	300
7	150

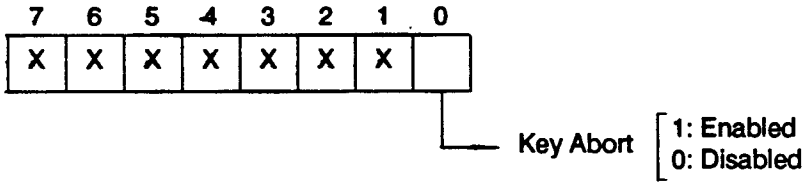


X = don't care

Figure L-3. HNBC Parity — Parameter Byte 3 (Offset 02h)

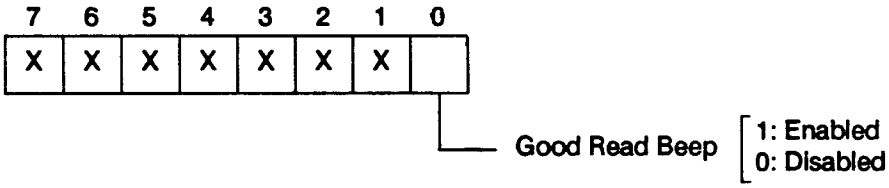
Table L-5. HNBC Parity Values

Value	Parity
0	Zero
1	One
2	Even
3	Odd



X = don't care

Figure L-4. HNBC Key Abort — Parameter Byte 4 (Offset 03h)



X = don't care

Figure L-5. HNBC Good Read Beep — Parameter Byte 5 (Offset 04h)

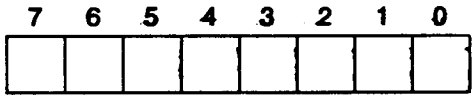


Figure L-6. HNBC Terminate Character * — Parameter Byte 6 (Offset 05h)

The default values for the parameters are FFh (valid data flag), 01h (9600 baud), 00h (zero parity), 01h (key abort enabled), 01h (good read beep enabled), and 00h (no terminate character).

HNSP Low-Level Handler for Serial Port

HNSP is a low-level bar code handler for the serial port. It is designed to allow "smart" bar code scanning devices to be connected to the serial port — devices which do on-board decoding of bar code labels into ASCII, and return it as serial data.

HNSP is designed to work with bar code devices whose electrical characteristics match those of the HP-94 serial port, and that send data in bursts of no more than 255 characters, with an intercharacter delay (time between characters) of 0-106 ms. HNSP is only supported for Hewlett-Packard Smart Wands (HP 39961D, HP 39963D, and HP 39965D).

HNSP Statistics

Here are pertinent statistics for HNSP.

Table L-6. HNSP Statistics

Item	Value
Version Number	1.00
Handler Identifier	SP
Valid Data Flag	FFh
Length in HP-94	2332 bytes
Scratch Areas Used	1 of 288 bytes *
Handler Information Table Offsets Used	02h
Valid Channel Numbers	1

* One additional 16-byte parameter scratch area is allocated if it was not allocated before opening the handler.

HNSP Capabilities

HNSP provides the following capabilities:

- Read/Write Operation
  Bar code data can be read from the port, and commands and data can be written to the bar code device. XON/XOFF handshaking is automatically used to pace transmission only.
- Good Read Beep
  Automatic beep on successful decoding of a bar code label.

* To disable use of the terminate character, set it to zero.

- Key Abort
  Allows a key being pressed to abort waiting for a successful scan.
- Received-Data Buffering
  Received data is placed in a 255-byte buffer. There is no transmit buffer.
- Speeds
  Speeds can be set from 150 to 9600 baud.
- Data Bits
  Seven only.
- Parity
  Zero, one, even, or odd parity.
- Stop Bits
  One for received data like HNBC. Two for transmitted data (actually, one plus intercharacter delay), which allows transmitting to devices that use either one or two stop bits.
- Terminate Character Control
  When defined, a received terminate character will signal the end of the bar code data from the scanning device. A terminate character will be sent after sending every block of data.

The table below describes how HNSP behaves. It shows the action taken by the handler routines as well as during its interrupt service routine, not including normal handler activities described in the "User-Defined Handlers" chapter. Note that certain actions, such as beeping on a good scan or responding to a received terminate character, will only occur if the appropriate options were enabled when the handler was opened.

Table L-7. Behavior of HNSP

Routine	Activities
CLOSE	Complete transmission of current byte Disable interrupt 53h and restore interrupt vector Lower RTS and DTR Wait 60 ms for signals to stabilize Disable 82C51 and turn off power to serial port Release handler scratch area
IOCTL	Implement the reserved IOCTL functions 00h-04h and 80h
OPEN	Allocate parameter scratch area if needed Allocate handler scratch area Take over interrupt 53h vector Enable 82C51 and supply power to serial port Initialize operating configuration * Raise RTS and DTR Return handler scratch area address in CX †
POWERON	Do nothing
READ	Wait for key up before accepting data Discard data until none for 106 ms to avoid reading middle of label Return no data and error 75h (117) if read aborted by pressing key Monitor and report low battery, power switch, and timeout errors ‡ Enable interrupt 53h Wait for first byte received End read operation if no subsequent data received for 106 ms Compute parity of received data Report error 74h (116) if terminate character detected Report error 75h (117) if scanned length less than requested length Report errors detected in interrupt service routine Issue high beep if scan successful Return data from receive buffer
RSVD2	Do nothing
RSVD3	Do nothing
TERM	Flush receive buffer
WARM	Perform all OPEN routine activities except those related to scratch areas
WRITE	Monitor and report low battery, power switch, and timeout errors Monitor CTS indirectly and report error DAh (218) if lost Write data to 82C51 Send terminate character at end of data
Interrupt Service	Monitor parity, framing, overrun, and receive buffer overflow errors Read data from 82C51 and accumulate data into receive buffer Disable/enable transmission when XOFF/XON received

* Baud rate, parity, key abort, good read beep, and terminate character.

† Handlers are not required to return this, but HNSP does.

‡ System timeout only monitored until first byte received. After that, no data received for 106 ms signals the end of the label. Consequently, all characters must be received in a burst in which the intercharacter delay (time between characters) must be less than 106 ms.

CAUTION While the READ routine is executing, the background timer routine (interrupt 1Ch) must not clear the CPU interrupt flag (CLI), write the interrupt control register (00h) to enable any interrupts, or issue software interrupts (such as interrupt 1Ah for the operating system functions). Doing so may cause loss of bar code data, resulting in parity or framing errors.

The errors reported by HNSP are shown in the following table.

Table L-8. Errors Reported by HNSP

Routine	Errors
CLOSE	6Eh
IOCTL	65h
OPEN	65h,67h,6Eh,71h
POWERON	None
READ	74h,75h,76h,77h,C8h,C9h,CAh,CBh,CCh,CDh,CEh,CFh,D0h
RSVD2	None
RSVD3	None
TERM	None
WARM	None
WRITE	76h,77h,C8h,DAh
Interrupt Service *	C9h,CAh,CBh,CCh,CDh,CEh,CFh,D0h
* Detected by interrupt service routine, but reported by READ routine.	

NOTE Two errors reported by the READ routine (74h, terminate character detected, and 75h, end of data) do not indicate error conditions, but signal the end of bar code data for the BASIC GET # and INPUT # statements. Assembly language programs using HNSP should handle these two errors differently than other errors from the READ routine.

If READ has not transferred all the data in its receive buffer when any read error occurs, it will flush the buffer. The next time READ is called, it will wait for a new bar code scan.

Parameters at OPEN Time

When HNSP is opened, it looks at offset 02h of the handler information table. If the value is zero, it allocates a one-paragraph parameter scratch area, places the default configuration in it, and places the scratch area address in the table. If the value is non-zero, it uses the value as the segment address of an existing parameter scratch area, and reads the configuration to use from that scratch area. The

meanings of the parameters are shown below. In these figures, the offsets are from the start of the parameter scratch area. A copy of these parameters are pointed to by `ES:DX` in the `GET_CONFIG` and `CHANGE_CONFIG` reserved `IOCTL` functions (01h and 02h).

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

Figure L-7. HNSP Valid Data Flag — Parameter Byte 1 (Offset 00h)

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

Baud Rate (see table for meaning)

Figure L-8. HNSP Baud Rate — Parameter Byte 2 (Offset 01h)

Table L-9. HNSP Baud Rate Values

Value	Baud Rate
1	9600
2	4800
3	2400
4	1200
5	600
6	300
7	150

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

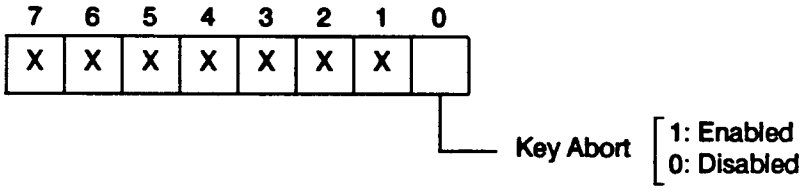
Parity (see table for meaning)

X = don't care

Figure L-9. HNSP Parity — Parameter Byte 3 (Offset 02h)

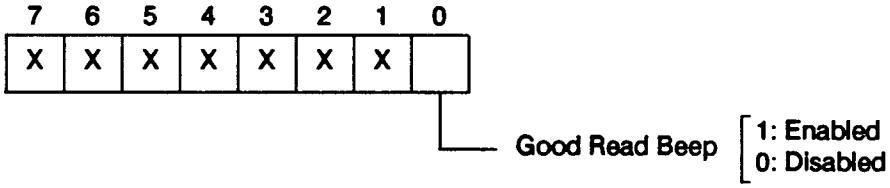
Table L-10. HNSP Parity Values

Value	Parity
0	Zero
1	One
2	Even
3	Odd



X = don't care

Figure L-10. HNSP Key Abort — Parameter Byte 4 (Offset 03h)



X = don't care

Figure L-11. HNSP Good Read Beep — Parameter Byte 5 (Offset 04h)

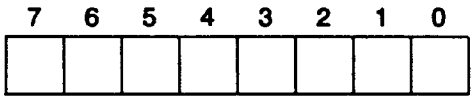


Figure L-12. HNSP Terminate Character * — Parameter Byte 6 (Offset 05h)

The default values for the parameters are FFh (valid data flag), 01h (9600 baud), 00h (zero parity), 01h (key abort enabled), 01h (good read beep enabled), and 00h (no terminate character).

* To disable use of the terminate character, set it to zero.

Write With Read Enabled IOCTL Function

HNSP implements an additional IOCTL function, function 80h, called WR_RD_EN (*write with read enabled*). It is invoked by setting AH to 80h when calling the HNSP IOCTL routine, and it returns 00h in AL (no errors). This is used when requesting status from a bar code device. To request status from a Hewlett-Packard Smart Wand, for example, a program would normally send it an escape sequence via the WRITE routine. The Smart Wand returns its status almost immediately — before the HNSP READ routine is ready to accept it. The READ routine will not read this status successfully because it ignores all data received after it is called until a quiet period of 106 ms has elapsed.

The WR_RD_EN function enables HNSP to receive data that arrives immediately after its WRITE routine completes writing data to the bar code device. The received data is stored in a buffer and returned to the calling program the next time the READ routine is called.

When the WR_RD_EN function is called, it enables a write with read enabled *only for the next write operation*. The next time the WRITE routine is called, it will actually do two separate I/O operations: first a write, then a read.

- The WRITE routine first performs the write operation the same way it would for a normal write. WRITE writes CX bytes of data starting at ES:BX out the serial port (for HP Smart Wands, this would be the status request escape sequence).
- The WRITE routine then performs the read operation the same way it would for a normal read. It waits until it receives the data from the serial port, or until the system timeout period expires. Once a byte has been received, the system timeout is no longer monitored, and WRITE assumes all data has arrived when the serial port does not receive any bytes for 106 ms.
- After all data has arrived, the WRITE routine checks for parity and framing errors. It does *not* beep, even if the beeper is enabled for normal bar code data. WRITE stores the received data in the receive buffer.
- WRITE returns the number of bytes actually written (*not* the number of bytes read) in CX and the error code in AL (00h if no errors). The error code is for both the write *and* read portions of the operation. The calling program will not know whether the error occurred during the write or the read, except for the context of the error message. For example, error DAh (218) can only occur during a write, while error CAh (202) can only occur during a read.

The next time the READ routine is called, it behaves as a normal read with data already available in the receive buffer, even though the data was actually received by the WRITE routine. The number of bytes actually read is returned in CX, and the data is returned in the read buffer specified by the READ caller.

The next time the WRITE routine is called, it behaves as a normal write — no write with read enabled operation will be performed unless WR_RD_EN is called again.

Because WR_RD_EN is treated as two separate I/O operations, the system timeout is restarted twice. It is started for the write operation and then stopped when the write is completed. It is then restarted for the read operation and stopped when the read is complete. If a system timeout occurs during either operation, AL is set to 76h (118). (For the read operation, it is only monitored until the first byte is received. After that, no data received for 106 ms signals the end of the label.)

XON/XOFF Handshaking During WR_RD_EN

XON/XOFF handshaking is normally done only during the WRITE routine. When the write with read enabled operation occurs, however, XON/XOFF handshaking is performed during both the write *and* the read portions of the operation. If status information returned by a bar code device contains an XON or an XOFF as legitimate data, those characters will be used to pace communications. They will not be passed back to the caller as part of the status.

HP Smart Wands do not send XON or XOFF characters as part of their status information.

HNWN High-Level Handler for Bar Code Handlers

HNWN is a high-level bar code handler for either the bar code port or the serial port. Because it is a high-level handler, it only communicates with low-level handlers, and specifically with HNBC and HNSP. It is designed to accommodate the unique features of Hewlett-Packard Smart Wands (HP 39961D, HP 39963D, and HP 39965D), and is only supported for these devices.

Throughout this section, there are references to features, behavior, and escape sequences sent or recognized by the Smart Wand. Refer to the *HP Smart Wand User's Manual* (part number HP 39960-90001) for details. There are also references to *configuration menus*, which provide optical configuration of the Smart Wand. This allows changing the Smart Wand's behavior by scanning bar code labels that are interpreted as commands, not as data. (Since the HP-94 bar code port is read-only, commands to change configuration cannot be sent to the Smart Wand through the bar code port). Refer to the *Smart Wand Configuration Menus* (part number HP 39960-90002) for details.

HNWN Statistics

Here are pertinent statistics for HNWN.

Table L-11. HNWN Statistics

Item	Value
Version Number	1.00
Handler Identifier	WN
Valid Data Flag	FFh
Length in HP-94	2217 bytes
Scratch Areas Used	1 of 272 bytes *
Handler Information Table Offsets Used	02h or 04h
Valid Channel Numbers	1 and 2
Valid Low-Level Handlers	HNBC, HNSP

* One additional 16-byte parameter scratch area is allocated if it was not allocated before opening the handler.

HNWN Capabilities

HNWN provides the following capabilities:

- Ignore or transmit Smart Wand escape sequences
  Causes escape sequences sent by the Smart Wand when it is in configuration mode to be sent to the calling program. Different beeps than for normal bar code data help distinguish received configuration escape sequences.
- Synchronize parity and baud rate of HP-94 port and Smart Wand
  Allows the HP-94 serial port or bar code port to track the Smart Wand's parity and baud rate without closing and reopening the port.

The table below describes how HNWN behaves. It shows the action taken by the handler routines, not including normal handler activities described in the "User-Defined Handlers" chapter. Note that certain actions, such as responding to escape sequences from the HNWN caller or from the Smart Wand, will only occur if the appropriate options were enabled when the handler was opened. Since high-level handlers interact with low-level handlers but not with I/O port hardware, HNWN has no interrupt service routine.

Table L-12. Behavior of HNWN

Routine	Activities
CLOSE	Call low-level handler CLOSE routine Release scratch area
IOCTL	Call low-level handler IOCTL routine
OPEN	Allocate parameter scratch area if needed Allocate handler scratch area Call low-level handler OPEN routine
POWERON	Do nothing
READ	Read data from low-level handler by calling its READ routine Ignore or transmit escape sequences
RSVD2	Call low-level handler RSVD2 routine
RSVD3	Call low-level handler RSVD3 routine
TERM	Call low-level handler TERM routine
WARM	Call low-level handler WARM routine
WRITE	Parse escape sequences being sent to low-level handler Take appropriate action for special escape sequences * Pass data to low-level handler by calling its WRITE routine

* Discussed later in this section.

NOTE HNWN cannot be used by itself — it must be used in conjunction with either HNBC or HNSP. To open HNWN with one of the low-level handlers, use the following as the handler name given to the BASIC OPEN # statement or the OPEN function (0Fh):

"HNWN;HNBC" for the bar code port (channel 2)

"HNWN;HNSP" for the serial port (channel 1)

When the low-level handlers are copied into the HP-94, their file names must be either of the low-level handlers must be either HNBC or HNSP — if the file names are different, HNWN will not be able to open them.

The errors reported by HNWN are shown in the following table. In addition, HNWN will report errors returned to it by either HNBC or HNSP.

Table L-13. Errors Reported by HNWN

Routine	Errors
CLOSE	6Eh
IOCTL	None
OPEN	65h,66h,67h,6Eh,71h
POWERON	None
READ	None
RSVD2	None
RSVD3	None
TERM	None
WARM	None
WRITE	None

Parameters at OPEN Time

When HNWN is opened, it looks in the handler information table. If it is opened to channel 1, it looks at offset 02h of the table. If it is opened to channel 2, it looks at offset 04h of the table. If the value is zero, it allocates a one-paragraph parameter scratch area, places the default configuration in it, and places the scratch area address in the table. If the value is non-zero, it uses the value as the segment address of an existing parameter scratch area, and reads the configuration to use from that scratch area. The meanings of the parameters are shown below. In these figures, the offsets are from the start of the parameter scratch area.

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

Figure L-13. HNWN Valid Data Flag — Parameter Byte 1 (Offset 08h)

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

Escape Sequences [1: Transmit / 0: Ignore]

Figure L-14. HNWN Escape Sequences — Parameter Byte 2 (Offset 09h)

The default values for the parameters are FFh (valid data flag) and 00h (ignore escape sequences).

Response to Escape Sequences From Smart Wand

When the Smart Wand scans bar codes in a configuration menu, it sends one of six types of responses:

- Configuration Complete (Ec *)
  This is sent to signify that the Smart Wand has completed the configuration operation specified by the menu.
- Configuration Partially Complete (Ec \ +)
  This is sent to signify the Smart Wand has completed a portion of the configuration operation. This is sent for intermediate steps in configuration operations that require more than one scan.
- Syntax Error (Ec \ -)
  This is sent to signify that the configuration menu was out of context. This may be caused by scanning configuration bar codes in the wrong order, that are the wrong type, or that are numerically out of range.
- Configuration Dump (Ec \ * CR LF Ec \ ...)
  This contains status information about the Smart Wand. If the Smart Wand is in HP-94 default mode, the length of this status is 223 characters.
- Hard Reset Message (`ready XX.X`)
  This message is sent if the configuration bar code that specifies a hard reset is scanned. `XX.X` is the Smart Wand's firmware version number.
- No Read Message (user-defined, default is CR LF)
  This message is sent only if the Smart Wand is enabled to send the no read message and if the Smart Wand reads a bar code label but is unable to decode it.

HNWN provides special responses only for the four escape sequences. It treats the hard reset message and no read message the same as standard bar code data. It is the responsibility of the calling program to provide special handling of these messages.

When escape sequences are received, HNWN will respond in one of two ways:

- Ignore Escape Sequences (default behavior)
  If this mode is selected, HNWN will discard all strings received from the Smart Wand that begin with Ec \. There is no beep, and the string is not passed to the calling program. This mode may be used if it is desirable to prevent configuration messages from accidentally being interpreted by an application as legitimate bar code data.

- Transmit Escape Sequences
  In this mode HNWN will transmit to the calling program all strings received from the Smart Wand that begin with Ec \. When escape sequences are received, HNWN causes the HP-94 to generate different sounding beeps in response to the configuration mode escape sequences. These are generated only if there are no parity or framing errors when the configuration bar code was scanned, and are generated whether or not the good read beep is enabled for normal bar code data.

Table L-14. Beeps From HNWN for Smart Wand Escape Sequences

Smart Wand Escape Sequence	Number of Beeps	Beep Tone
Configuration Complete	4	High
Configuration Partially Complete	2	High
Syntax Error	4	Low

NOTE Because Code 128 bar code labels can contain any of 128 ASCII characters, it is possible (although unlikely) to encounter Code 128 labels that decode to strings beginning with Ec \. If such labels are encountered, HNWN will respond to them as if they were configuration sequences (assuming the transmit escape sequences option is being used). Applications that may encounter this situation should use HNWN with the ignore escape sequences option, or use HNBC or HNSP alone (without HNWN).

Response to Escape Sequences From Calling Program

The Smart Wand will respond to a number of escape sequences sent to it through its serial port (using HNSP). Four of these also invoke special responses from HNWN:

- Serial Port Configuration (Ec - y *n* p)
- Status Request (Ec - y *n* s)
- Hard Reset (Ec - y 1 z and Ec E)
- Save Configuration to Non-Volatile Memory (Ec - y 5 z)

These may be sent to HNWN ; HNSP with the BASIC PRINT # and PUT # statements or with the WRITE function (13h). The last three cannot be sent to the Smart Wand using HNBC since the bar code port is read-only (the first one is handled by HNWN ; HNBC as a special case). Refer to the "Hardware Specifications" for the pin assignments of a cable that will connect the serial port to the Smart Wand.

Serial Port Configuration Escape Sequence

The format of this escape sequence is as follows:

$$^{E}c\text{-}y\,n\,p$$

where *n* is a sequence of numeric characters (30h through 39h) that specifies a decimal number between 0 and 255. If this decimal number is converted to the equivalent binary number, the bit pattern has the following meaning:

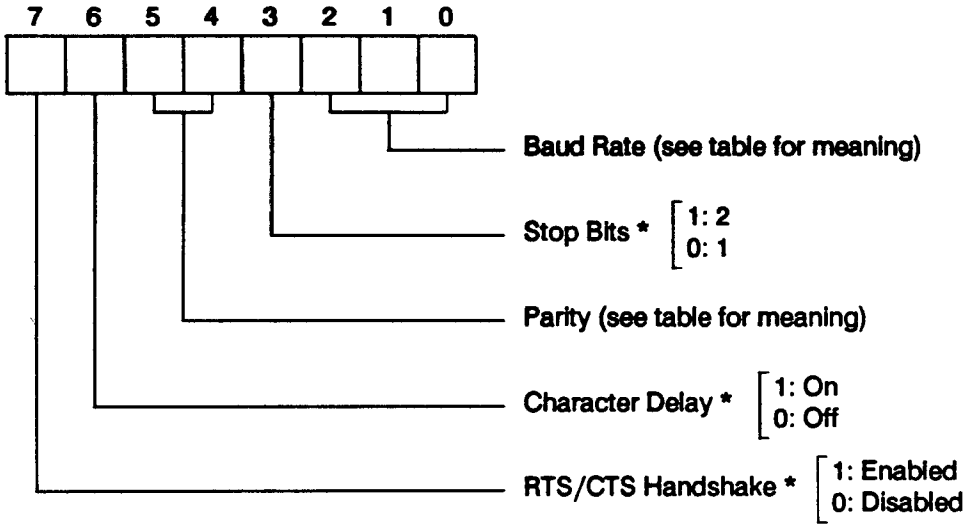


Figure L-15. Serial Port Configuration Escape Sequence

Table L-15. Smart Wand Baud Rate Values

Value	Baud Rate
0	150
1	300
2	600
3	1200
4	2400
5	4800
6	9600

* Ignored by HNWN. Only affects Smart Wand.

Table L-16. Smart Wand Parity Values

Value	Parity
0	Zero
1	One
2	Even
3	Odd

If the Smart Wand receives this escape sequence through its serial port, it changes its serial configuration as specified. For example, Ec - y 62 p would set the Smart Wand serial port to 9600 baud, 2 stop bits, odd parity, character delay off, and RTS/CTS handshake disabled (because 62 decimal corresponds to a bit pattern of 00111110).

The manner in which HNWN responds depends on which channel it is open to. If it is open to the serial port (channel 1) through HNSP, HNWN sends the escape sequence on to the Smart Wand at the current baud rate and parity. It then changes the baud rate and parity of the HP-94 serial port to the values specified by the sequence. This causes the Smart Wand and the HP-94 to track each other's serial configuration. When the port configuration is changed this way, the new configuration is assumed only for as long as the port is open. If the bar code or serial port is closed and then reopened, it will assume the baud rate and parity specified in the parameter scratch area at the time the port is reopened.

If HNWN is open to the bar code port (channel 2) through HNBC, HNWN changes the baud rate and/or the parity of the port. It does not try to write the escape sequence to the bar code port, since the port is read-only. (For this reason, the serial port configuration escape sequence is the only sequence that may be written to HNWN ; HNBC without causing an error.) It is assumed that immediately after this sequence is sent to HNWN ; HNBC, the operator will scan a configuration bar code that causes the Smart Wand to change parity and baud rate to match those of the HP-94. If this is not done, all subsequent scans will result in parity or framing errors.

If it is desirable to change both baud rate and parity for the bar code port, the baud rate should be changed first. The changes should be done as two separate operations, since each change involves sending an escape sequence to HNWN and having the operator scan the appropriate configuration bar code.

When the Smart Wand is powered off, then back on, it may or may not return to its default serial configuration. This depends on whether the Smart Wand's serial port configuration has been saved (discussed in "Save Configuration to Non-Volatile Memory Escape Sequence").

Status Request Escape Sequence

This escape sequence can be used to obtain various types of status from the Smart Wand. The format is as follows:

Ec - y *n* s

where *n* is a decimal number from 1-6. The meanings of the different values of *n* are as follows:

Table L-17. Status Request Escape Sequence Parameter

Value	Type of Status Returned
1	Status message followed by ^{C}R
2	Status message with selected trailer
3	Message ready/not ready response (for Single Read Mode 2)
4 *	Smart Wand configuration screen message
5	Serial number
6	Configuration dump †

* The Smart Wand responds to ^{E}C - y 4 s by sending its configuration screen message, which is not usable by the HP-94. HNWN traps this sequence, does not send it to the Smart Wand, and returns error 65h (101).

† If the Smart Wand is in HP-94 default mode, this is 223 bytes long.

If this escape sequence is written to channel 1, it alters the behavior of HNWN;HNSP. Normally, HNWN;HNSP discards all data received by the serial port unless its READ routine is called. However, when this escape sequence is received, HNWN invokes the WR_RD_EN function of the HNSP IOCTL routine, writes the escape sequence to HNSP, then places the status returned by the Smart Wand in the receive buffer. The status will be returned to the calling program the next time the READ routine is called. The beeper does not sound when the status information is received, even if beeps are enabled for normal bar code data.

If this escape sequence is written to channel 2, HNWN;HNBC returns error 6Dh (109).

NOTE The status messages returned by the Smart Wand are escape sequences, and HNWN must be configured to transmit escape sequences in order for the calling program to receive the status messages.

Wand Hard Reset Escape Sequences

There are two escape sequences that cause the Smart Wand to perform a hard reset. These are:

^{E}C - y 1 z

^{E}C E

When the Smart Wand receives one of these escape sequences, it becomes unable to parse escape sequences for 516 ms (worst case), until the reset operation is complete. When HNWN receives either of these escape sequences, it sends it to the Smart Wand (through HNSP only) and then waits 530 ms before returning to the calling program or sending any more characters in the output string. This gives the Smart Wand enough time to perform the reset operation.

Save Configuration to Non-Volatile Memory Escape Sequence

This escape sequence has the following format:

^{E}c - y 5 z

It causes the Smart Wand to write its current configuration to the Smart Wand's built-in non-volatile memory (EEPROM). This operation requires 2.78 seconds (worst case). When HNWN receives this escape sequence, it sends it to the Smart Wand (through HNSP only) and then waits 2.9 seconds before returning to the calling program or sending any more characters in the output string. The HP-94 power switch is disabled during this period to prevent powering down of the HP-94 and the Smart Wand.

CAUTION The Smart Wand must not be powered down by turning off the HP-94 while the save configuration operation is in progress. Although the power switch is disabled, the reset switch or automatic turn off after very low battery could still turn the 94 and Smart Wand off. If this occurs, the Smart Wand may become inoperable, requiring that it be sent to a Hewlett-Packard service center to be restored to proper operation.

M

Disc-Based Utility Routines

The disc included with the *HP-94 Technical Reference Manual* contains 17 utility routines. These utilities are include files with the extension ASM. They can be included as part of assembly language programs (using the INCLUDE assembler directive), and can be executed from either RAM or ROM. Below is a list of all the utilities.

Table M-1. Utility Routines on Technical Reference Manual Disc

File Name	Description	Page
BLINK.ASM *	Blink the cursor	M-3
EQUATES.ASM	Equates for HP-94 operating system	M-5
FINDOS.ASM	Locate operating system file in system ROM	M-8
INTERNAL.ASM	Call internal entry point of BASIC keyword	M-10
IOABORT.ASM	Check for low battery, power switch, or timeout	M-14
IOWAIT.ASM	Enable I/O wait state	M-18
ISOPEN.ASM	Determine if a channel is open	M-20
LLHLINKG.ASM	Call low-level handler from high-level handler	M-22
NOIOWAIT.ASM	Disable I/O wait state	M-34
READCTRL.ASM	Examine hardware status	M-36
READINTR.ASM	Examine interrupt status	M-38
SCANKYBD.ASM *	Check if key down	M-40
SETCTRL.ASM	Write to saved copy of main control register	M-42
SETINTR.ASM	Write to saved copy of interrupt control register	M-44
VERSION.ASM *	Return version of operating system or program	M-46
XIOCTL.ASM	Execute IOCTL routine in any handler	M-49
XTIMEOUT.ASM *	Execute timeout process when timeout occurs	M-51

* Requires the FINDOS.ASM utility.

These utilities were written to be assembled using the Microsoft assembler MASM. Conditional assembly is used to allow the Hewlett-Packard copyright notice to appear in the source code, but not be printed in the list file (extension LST). The copyright notice allows the utilities to be reproduced for inclusion in an application or for archival purposes *without* prior written consent of Hewlett-Packard.

Conditional assembly is also used for the FINDOS utility. This utility is required by the BLINK, SCANKYBD, VERSION and XTIMEOUT utilities, and is included with each of them (using the INCLUDE assembler directive). The conditional assembly prevents FINDOS from being included more than once in the source file.

Utility Routine Descriptions

Utility routine descriptions consist of the following:

- A brief description of the utility.
- Information on when the utility should be used.
- Program listing.

The program listings start with a comment block that describes the following:

- What the utility does.
- How to call the utility.
- What is returned by the utility.
- Registers altered by the utility.

The BLINK utility in this include file blinks the cursor. Normally, the system timer interrupt service routine causes the cursor to blink every 500 ms. However, in time-critical handlers such as for the bar code port, the system timer may be disabled while waiting for data to be received at the port (to prevent bar code port transition interrupts from being missed). BLINK performs the operating system cursor blink operation when the system timer is disabled.

The BLINK utility should be called approximately every 100 ms while the handler waits for data, thereby allowing the cursor to continue blinking. This helps prevent users from gaining the perception of no machine activity that accompanies an idle cursor. (The 100 ms calling interval is consistent with the frequency with which the system timer interrupt routine calls BLINK.)

BLINK uses FINDOS to find the operating system file and the start of the operating system jump table.

Program listing:

```
                        .sfcond
                        if1
;*************************************************************
;*     "(c) Copyright Hewlett-Packard Company, 1987.   All   *
;*     rights are reserved.  Copying or other reproduction   *
;*     of this program for inclusion in an application or    *
;*     for archival purposes is permitted without the prior  *
;*     written consent of Hewlett-Packard Company."          *
;*************************************************************
                        endif
                        .lfcond

                        include findos.asm

;*************************************************************
;*
;* Name: BLINK
;*
;* Version: 1.3
;*
;* Description:
;*   Call the cursor blink routine in the operating system
;*
;* Call with:
;*   None
;*
;* Returns:
;*   None
;*
;* Registers altered:
;*   None
;*
;* Notes:
;*   The BLINK routine is normally called every 100 ms by the
;*   system timer interrupt service routine, and should be called
;*   here only when the system timer interrupt is disabled.
;*
;*   The BLINK routine decrements a count in the operating system
;*   scratch space each time it is called.  The cursor state is
;*   changed only when the count reaches 0. When the cursor state
```

```
;*   is changed, the count is reset to 5.
;*
;*****************************************************************


BLINK                   proc        near
                        push        bx
                        push        si
                        push        ds
                        push        es
                        call        FINDOS
; DS is SYOS segment
; ES is operating system pointer table segment
                        push        ds                  ; Push cursor blink routine segment
                        push        es:[38h]            ; Push cursor blink routine offset
                        mov         si,es:[00h]
                        mov         ds,si               ; DS = operating system data segment
                        mov         si,sp
                        call        dword ptr ss:[si]
                        add         sp,4
                        pop         es
                        pop         ds
                        pop         si
                        pop         bx
                        ret
BLINK                   endp
```

EQUATES.ASM

The EQUATES include file is a set of symbolic names for use in writing HP-94 assembly language programs. These include names for operating system functions, register locations in the register save area for handler routines, and certain operating system values and locations.

Program listing:

```
                        .sfcond
                        if1
;*****************************************************************
;*      "(c) Copyright Hewlett-Packard Company, 1986, 1987.  All *
;*      rights are reserved.  Copying or other reproduction of   *
;*      this program for inclusion in an application or for      *
;*      archival purposes is permitted without the prior written *
;*      consent of Hewlett-Packard Company."                     *
;*****************************************************************
                        endif
                        .lfcond

; Equate values for the HP-94

; Macros to push and pop registers in order expected by handlers
pushregs                macro
                        pushf
                        push    bp
                        push    es
                        push    ds
                        push    di
                        push    si
                        push    dx
                        push    cx
                        push    bx
                        push    ax
                        mov     bp,sp
                        endm

popregs                 macro
                        pop     ax
                        pop     bx
                        pop     cx
                        pop     dx
                        pop     si
                        pop     di
                        pop     ds
                        pop     es
                        pop     bp
                        popf
                        endm

; Symbolic names for registers relative to BP
; (e.g. AXREG[BP] is the saved value of AX)

AXREG                   equ     00h
ALREG                   equ     00h
AHREG                   equ     01h
;
BXREG                   equ     02h
BLREG                   equ     02h
BHREG                   equ     03h
```

```
;
CXREG                   equ         04h
CLREG                   equ         04h
CHREG                   equ         05h
;
DXREG                   equ         06h
DLREG                   equ         06h
DHREG                   equ         07h
;
SIREG                   equ         08h
DIREG                   equ         0Ah
DSREG                   equ         0Ch
ESREG                   equ         0Eh
BPREG                   equ         10h


OS_PTRTBL_SEG           equ         16h                 ; Operating system pointer table segment
;
; HP-94 ADDRESSES
;
; OS_PTRTBL_SEG is a segment address of a table of operating system pointers.
; The contents of this table point to where the system addresses
; are located.  All entries in the address table are 2-byte entries.
; The values below are the offset addresses in OS_PTRTBL_SEG.
;
OSSCRATCH_SEG           equ         00h                 ; System RAM data segment
;
SYSROM_SEG              equ         08h                 ; System ROM segment
;
OPENTBLSIZE             equ         0Ah                 ; Number of open table entries
;
RESTART_STATUS          equ         14h                 ; Offset of status area in OSRAM_SEG
;
MAXDIR                  equ         18h                 ; Offset of maximum directory # in OSRAM_SEG
EVENTTBL                equ         1Ah                 ; Offset of system timer event table in OSRAM_SEG
;
OPENTBL                 equ         24h                 ; Offset of channel # table in OSRAM_SEG
;
KEY_SCAN                equ         36h                 ; Offset (in "SYOS" file) of jump to key scan routine
CURSOR_BLINK            equ         38h                 ; Offset (in "SYOS" file) of jump to cursor blink rout
TIMEOUT                 equ         3Ah                 ; Offset (in "SYOS" file) of jump to timeout utility
VERSION                 equ         3Ch                 ; Offset (in "SYOS" file) of system ROM version number

;
; FUNCTION CALLS
;
; Each function call equate has two forms, one for directly loading
; AH (mov  ah,FUNCTION), the other for loading AX with a word value.
; The form for loading AX has "x100h" appended to the base name.
;
; Call operating system functions as follows:
;
;  Example which loads AH:
;    Output the line pointed to by ES:BX
;
;                        mov         ah,PUT_LINE
;                        int         1Ah
;
END_PROGRAM             equ         00h
GET_CHAR                equ         01h
```

```
GET_LINE                equ         02h
PUT_CHAR                equ         03h
PUT_LINE                equ         04h
CURSOR                  equ         05h
BUFFER_STATUS           equ         06h
BEEP                    equ         07h
TIME_DATE               equ         08h
TIMEOUT                 equ         09h
SET_INTR                equ         0Ah
GET_MEM                 equ         0Bh
REL_MEM                 equ         0Ch
MEM_CONFIG              equ         0Dh
ROOM                    equ         0Eh
OPEN                    equ         0Fh
CLOSE                   equ         10h
CREATE                  equ         11h
READ                    equ         12h
WRITE                   equ         13h
DELETE                  equ         14h
SEEK                    equ         15h
FIND_FILE               equ         16h
FIND_NEXT               equ         17h
DISPLAY_ERROR           equ         18h
;
;  Example which loads AX:
;    Output the letter 'j' to the LCD
;
;                       mov         ax,PUT_CHARx100h + "j"
;                       int         1Ah
;
END_PROGRAMx100h        equ         0000h
GET_CHARx100h           equ         0100h
GET_LINEx100h           equ         0200h
PUT_CHARx100h           equ         0300h
PUT_LINEx100h           equ         0400h
CURSORx100h             equ         0500h
BUFFER_STATUSx100h      equ         0600h
BEEPx100h               equ         0700h
TIME_DATEx100h          equ         0800h
TIMEOUTx100h            equ         0900h
SET_INTRx100h           equ         0A00h
GET_MEMx100h            equ         0B00h
REL_MEMx100h            equ         0C00h
MEM_CONFIGx100h         equ         0D00h
ROOMx100h               equ         0E00h
OPENx100h               equ         0F00h
CLOSEx100h              equ         1000h
CREATEx100h             equ         1100h
READx100h               equ         1200h
WRITEx100h              equ         1300h
DELETEx100h             equ         1400h
SEEKx100h               equ         1500h
FIND_FILEx100h          equ         1600h
FIND_NEXTx100h          equ         1700h
DISPLAY_ERRORx100h      equ         1800h
```

FINDOS.ASM

The FINDOS utility in this include file finds the operating system (file SYOS) in the system ROM. The FIND_FILE and FIND_NEXT functions (16h and 17h) cannot be used because running programs do not have access to directory 5, the system ROM directory. The FINDOS utility searches the system ROM directory table to locate SYOS.

This utility is used by the BLINK, SCANKYBD, and XTIMEOUT utilities, all of which utilities call routines whose locations are defined by a jump table at a known location in the operating system file. It is also used by VERSION to locate the version number at a known location.

Program listing:

```
                        .xlist      ; Suppress findoshere macro listing
findoshere              macro
                        .sfcond
                        if1
;*************************************************************
;*     "(c) Copyright Hewlett-Packard Company, 1987.   All   *
;*     rights are reserved.  Copying or other reproduction   *
;*     of this program for inclusion in an application or    *
;*     for archival purposes is permitted without the prior  *
;*     written consent of Hewlett-Packard Company."          *
;*************************************************************
                        endif
                        .lfcond

TABLE_SEG               EQU         16h
SYSROM_SEG              EQU         08h
;*********************************************************
;*
;* Name: FINDOS
;*
;* Version: 1.3
;*
;* Description:
;*   Find the start of the SYOS file
;*
;* Call with:
;*   None
;*
;* Returns:
;*   DS = start of SYOS file (0 if SYOS not found)
;*   ES = start of operating system pointer table
;*
;* Registers altered:
;*   DS, ES
;*
;* Notes:
;*   If SYOS is not found, DS = 0.
;*
;*********************************************************

FINDOS                  proc        near
                        push        bx
                        push        cx

                        mov         bx,TABLE_SEG
                        mov         es,bx               ; ES is TABLE_SEG
```

```
                mov     bx,es:[SYSROM_SEG]
                mov     ds,bx               ; DS is SYSROM_SEG
                mov     cx,ds:[06h]         ; Get start of files pointer
                sub     cx,bx               ; CX is number of paragraphs in directory
                dec     cx                  ; Account for "*DIR*" entry
FIND1:
                mov     bx,ds
                inc     bx
                mov     ds,bx               ; DS[0] is name
                cmp     ds:[2],'O'+'S'*100h ; 'OS'
                jne     FIND2
                cmp     ds:[0],'S'+'Y'*100h ; 'SY'
                je      FIND3
FIND2:
                loop    FIND1
NOFIND:
                sub     bx,bx
                mov     ds,bx               ; Set DS = 0 (not found)
                jmp     short FIND4
;-
FIND3:
                mov     ds,ds:[7]           ; DS is SYOS segment
FIND4:
                pop     cx
                pop     bx
                ret
FINDOS          endp
                endm
                .list
                ifndef  FINDOS
                findoshere
                endif
                if      FINDOS eq $
                findoshere
                endif
```

INTERNAL.ASM

The INTERNAL utility in this include file calls the internal entry point of a type A file. The internal entry point is the address at offset 02h in the file — the second pair of bytes in the program header. It is used mainly for type A files that are new BASIC keywords, allowing access to the functionality of the keyword without using the interaction between the keyword and the BASIC interpreter. Refer to the "Program Execution" chapter in part 1, "Operating System", for details.

The INTERNAL utility calls the internal entry point with a FAR CALL, so the called program should end with a FAR RET.

Program listing:

```
                        .sfcond
                        if1
;*************************************************************
;*     "(c) Copyright Hewlett-Packard Company, 1987.   All   *
;*     rights are reserved.  Copying or other reproduction   *
;*     of this program for inclusion in an application or    *
;*     for archival purposes is permitted without the prior  *
;*     written consent of Hewlett-Packard Company."          *
;*************************************************************
                        endif
                        .lfcond

;*************************************************************
;*
;* Name: INTERNAL
;*
;* Version: 1.3
;*
;* Description:
;*   Call the internal entry point of a type "A" file
;*
;* Call with:
;*   SS:SP+2 = segment address of file name
;*   SS:SP = offset address of file name
;*
;* Returns:
;*   AL = Error code:
;*        00h       No error
;*        65h (101) Illegal parameter
;*        66h (102) Invalid directory number
;*        67h (103) File not found
;*
;* Registers altered:
;*   AL (if the internal entry point is called, the return
;*   value in AL is the value returned by the internal entry
;*   point)
;*
;* Notes:
;*   INTERNAL verifies that the file is type "A", and that
;*   the internal entry point offset is within the file.
;*
;*   All registers passed to INTERNAL are preserved for the
;*   call to the internal entry point.
;*
;*   The address of the file name is passed on the stack so that
;*   all registers may be passed to the internal entry point routine
```

```
;*
;*****************************************************

FIND_FILE           equ     16h
BUFFER_SIZE         equ     0Eh

AXREG               equ     00h
ALREG               equ     00h
AHREG               equ     01h
;
BXREG               equ     02h
BLREG               equ     02h
BHREG               equ     03h
;
CXREG               equ     04h
CLREG               equ     04h
CHREG               equ     05h
;
DXREG               equ     06h
DLREG               equ     06h
DHREG               equ     07h
;
SIREG               equ     08h
DIREG               equ     0Ah
DSREG               equ     0Ch
ESREG               equ     0Eh
BPREG               equ     10h
;
FLAGREG             equ     12h
;
REGSAVE_SIZE        equ     14h
FILE_SEGMENT        equ     REGSAVE_SIZE+BUFFER_SIZE+4
FILE_OFFSET         equ     REGSAVE_SIZE+BUFFER_SIZE+2

pushregs            macro
                    pushf
                    push    bp
                    push    es
                    push    ds
                    push    di
                    push    si
                    push    dx
                    push    cx
                    push    bx
                    push    ax
                    mov     bp,sp
                    endm

popregs             macro
                    pop     ax
                    pop     bx
                    pop     cx
                    pop     dx
                    pop     si
                    pop     di
                    pop     ds
                    pop     es
                    pop     bp
                    popf
                    endm
```

```
INTERNAL                proc        near
                        sub         sp,BUFFER_SIZE      ; Reserve file information buffer
;
                        pushregs
                        lea         dx,[bp+REGSAVE_SIZE]; offset of file information buffer
;
                        push        ss
                        pop         ds                  ; DS = SS
                        mov         es,FILE_SEGMENT[bp]
                        mov         bx,FILE_OFFSET[bp]
; DS:DX = address of file information buffer
                        mov         ah,FIND_FILE
                        int         1Ah                 ; O.S. function call
; Check for errors...
                        or          al,al
                        jnz         ENTRY_ERROR
; CX:DX = directory table entry of the file
                        push        ss
                        pop         ds
                        mov         si,sp
                        lea         si,[si+REGSAVE_SIZE]
; DS:SI = directory table entry of the file
                        mov         al,ds:[si+07h]
                        cmp         al,"A"
                        jne         ENTRY_NOT_A
                        mov         cx,ds:[si+0Ch]      ; high byte of end-of-data address
                        mov         dx,ds:[si+0Ah]      ; low word of end-of-data address
                        mov         ds,ds:[si+08h]      ; segment address of file
                        mov         ax,ds:[02h]         ; Internal entry point offset
                        cmp         ax,06h              ; Check for valid offset (must be >= 6)
                        jb          ENTRY_BAD
                        or          cx,cx
                        jnz         ENTRY_CALL
                        cmp         ax,dx               ; low word of end-of-data address
                        jae         ENTRY_BAD
; Address is OK
; DS is segment of file
ENTRY_CALL:
                        mov         FILE_SEGMENT[bp],ds ; segment of internal entry point
                        mov         FILE_OFFSET[bp],ax  ; offset of internal entry point
                        popregs                         ; restore all registers
                        add         sp,BUFFER_SIZE      ; discard buffer (no longer needed)
                        push        cs
                        push        sp                  ; leave room for offset of INTERNAL1
                        pushf
                        sub         sp,4                ; leave room for segment and offset addresses
                        push        bp
                        mov         bp,sp
                        push        ax
;
; Stack relative to BP (* means not yet filled in)
;
; 10h                   FILE_SEGMENT
; 0Eh                   FILE_OFFSET
; 0Ch                   Caller's return address (offset)
; 0Ah                   CS (my segment)
; 08h                   * offset of INTERNAL1
; 06h                   Flag register
; 04h                   * Segment of internal entry point
; 02h                   * Offset of internal entry point
```

```
; 00h                   my BP
;-02h                   my AX
;
                        mov     ax,offset INTERNAL1
                        mov     [bp+08h],ax
                        mov     ax,[bp+10h]         ; file segment address
                        mov     [bp+4],ax
                        mov     ax,[bp+0Eh]         ; file offset address
                        mov     [bp+2],ax
                        pop     ax
                        pop     bp
                        iret                        ; Internal entry point ends with a FAR RET
INTERNAL1:
                        ret     4                   ; NEAR RET and add 4 to SP
;-
ENTRY_NOT_A:
ENTRY_BAD:
                        mov     al,65h              ; Illegal parameter
ENTRY_ERROR:
                        mov     bp,sp
                        mov     ALREG[bp],al
                        popregs
                        add     sp,BUFFER_SIZE
                        ret     4                   ; NEAR RET and add 4 to SP

INTERNAL                endp
```

IOABORT.ASM

The IOABORT utility in this include file allows a handler to check for system errors that should cause I/O to be aborted: low battery, power switch pressed, and system timeout. IOABORT will report errors C8h (200), 77h (119), and 76h (118) respectively for these conditions, but only if the operating system I/O wait state has been enabled using IOWAIT. To use this during the READ or WRITE routine, the handler would do the following:

- Enable the operating system I/O wait state by calling IOWAIT.
- Call IOABORT periodically while waiting to receive or transmit data, and check if it returns an error code that indicates I/O should be aborted. It can be called as often as is convenient, such as in the main READ or WRITE routine wait loop. It should be called at least every second, since that is the system timeout resolution (although low battery or power switch may not occur exactly on a 1 second time boundary).
- If the timeout error is reported by IOABORT, the user-defined timeout interrupt routine defined by SET_INTR (0Ah) will *not* have been executed. The handler should call XTIMEOUT which will call the user-defined timeout interrupt routine if one was defined, or turn the machine off. If the low battery or power switch errors are reported by IOABORT, user-defined low battery or power switch interrupt routines defined by SET_INTR (0Ah) will *already* have been executed.
- Abort I/O by halting the process of receiving or sending data.
- Disable the operating system I/O wait state by calling NOIOWAIT.
- End the READ or WRITE routine, and return the error code from IOABORT to the caller.

IOABORT must be used in conjunction with IOWAIT, which sets the operating system I/O wait state. The tables below show how the I/O wait state affects how each of these error conditions are reported by IOABORT.

Table M-2. Low Battery Interrupt Routine Behavior During I/O

I/O Wait State	User-Defined Behavior	Default Behavior
Waiting *	IOABORT reports error C8h (200). User-defined low battery interrupt routine executed when low battery condition occurs. †	Program halted, Error 200 displayed, and machine waits for power switch to be pressed to turn off.
Not waiting	IOABORT does not report an error. User-defined low battery interrupt routine executed when low battery condition occurs.	Program halted, Error 200 displayed, and machine waits for power switch to be pressed to turn off.

* Only if IOWAIT was called.

† Routine has already been executed by the time IOABORT reports the error.

Table M-3. Power Switch Interrupt Routine Behavior During I/O

I/O Wait State	User-Defined Behavior	Default Behavior
Waiting *	IOABORT reports error 77h (119). User-defined power switch interrupt routine executed when power switch pressed. † Error not reported and interrupt routine not called if power switch disabled.	Machine turns off. No default action taken if power switch disabled.
Not waiting	IOABORT does not report an error. User-defined power switch interrupt routine executed when power switch pressed. Interrupt routine not called if power switch disabled.	Machine turns off.

* Only if IOWAIT was called.

† Routine has already been executed by the time IOABORT reports the error.

Table M-4. Timeout Interrupt Routine Behavior During I/O

I/O Wait State	User-Defined Behavior	Default Behavior
Waiting *	IOABORT reports error 76h (118). Handler must call XTIMEOUT, which will execute user-defined timeout interrupt routine. Error not reported if timeout disabled.	IOABORT reports error 76h (118). Handler must call XTIMEOUT, which will turn machine off. Error not reported if timeout disabled.
Not waiting	IOABORT does not report an error. User-defined timeout interrupt routine not executed.	IOABORT does not report an error. No default action taken.

* Only if IOWAIT was called.

Program listing:

```
                        .sfcond
                        if1
;*************************************************************
;*     "(c) Copyright Hewlett-Packard Company, 1986.  All    *
;*     rights are reserved.  Copying or other reproduction   *
;*     of this program for inclusion in an application or    *
;*     for archival purposes is permitted without the prior  *
;*     written consent of Hewlett-Packard Company."          *
;*************************************************************
                        endif
                        .lfcond

;*************************************************************
```

```
;*
;* Name: IOABORT
;*
;* Version: 1.3
;*
;* Description:
;*   Check for any error conditions while waiting for data
;*   (designed for use by a handler while doing I/O)
;*
;* Call with:
;*   None
;*
;* Returns:
;*   AL = Error code:
;*        00h        No error
;*        76h (118)  Timeout
;*        77h (119)  Power switch pressed
;*        C8h (200)  Low battery
;*
;* Registers altered:
;*   AL
;*
;* Notes:
;*   Timeout is checked first, followed by low battery,
;*   and finally power switch pressed; if there are multiple
;*   error conditions, only the first one found will be
;*   reported.  Subsequent calls to IOABORT will report
;*   any errors not previously reported.
;*
;*   IOABORT assumes that IOWAIT and NOIOWAIT are used
;*   by the handler to set up timeout processing.
;*
;*****************************************************************

IOABORT                 proc        near

                        push        bx
                        push        cx
                        push        ds
                        mov         bx,16h
                        mov         ds,bx
                        mov         bx,ds:[14h]
                        add         bx,4                ; BX points to current status
                        mov         ds,ds:[00h]
                        mov         al,ds:[bx]
                        mov         cx,0FE76h
                        test        al,01H              ;timeout?
                        jne         IOABORT1            ;(yes)
                        mov         cx,0F7C8h
                        test        al,08h              ;low battery?
                        jne         IOABORT1            ;(yes)
                        mov         cx,0EF77h
                        test        al,10H              ;power SW off?
                        jne         IOABORT1            ;(yes)
                        mov         cx,0FF00h
IOABORT1:
                        and         byte ptr ds:[bx],ch ;clear flag with CH
                        mov         al,cl
                        pop         ds
                        pop         cx
```

```
                        pop     bx
                        ret
IOABORT                 endp
```

IOWAIT.ASM

The IOWAIT utility in this include file enables the operating system I/O wait state. This is the state in which low battery, power switch, and timeout errors can be reported by the IOABORT utility while handler READ or WRITE routines are waiting for I/O.

The low battery error will occur when the operating voltage drops to 4.6 ± 0.05 volts or below. The power switch error will occur when the power switch is pressed. It will not occur if the power switch has been disabled using the SET_INTR function (0Ah). The timeout error will occur when the current system timeout value expires. The system timeout value is set by the TIMEOUT function (09h), and has a default time of 120 seconds. The timeout error will not occur if the timeout has been disabled by setting it to zero.

Whenever IOWAIT is called, it resets the timeout to the system timeout value. This allows a handler to restart the timeout period after each byte is sent or received by calling IOWAIT again.

When a handler READ or WRITE routine ends, it must call NOIOWAIT to indicate that I/O is not waiting.

The I/O wait state set by IOWAIT determines how IOABORT reports these error conditions. Refer to the IOABORT utility for details.

Program listing:

```
                        .sfcond
                        if1
;*************************************************************
;*     "(c) Copyright Hewlett-Packard Company, 1986.   All   *
;*     rights are reserved.  Copying or other reproduction   *
;*     of this program for inclusion in an application or    *
;*     for archival purposes is permitted without the prior  *
;*     written consent of Hewlett-Packard Company."          *
;*************************************************************
                        endif
                        .lfcond

;*************************************************************
;*
;* Name: IOWAIT
;*
;* Version: 1.4
;*
;* Description:
;*   Enable I/O wait state
;*
;* Call with:
;*   None
;*
;* Returns:
;*   None
;*
;* Registers altered:
;*   None
;*
;* Notes:
;*   Enables timeout if timeout interval is not zero.
```

```
;*
;*************************************************************

IOWAIT          proc    near
                push    ax
                push    bx
                push    si
                push    di
                push    bp
                push    ds
                mov     ax,16h
                mov     ds,ax
                mov     bx,ds:[14h]
                add     bx,4                ; BX points to current status
                mov     si,ds:[1Ah]         ; System timer event control table
                mov     di,ds:[18h]
                mov     bp,di
                add     di,2                ; DI points to timeout interval
                add     bp,0bh              ; BP points to timeout counter
                mov     ds,ds:[00h]         ; DS = operating system data segment
                or      byte ptr ds:[bx],20h; I/O waiting flag set
                mov     ax,word ptr ds:[di] ; Read timeout interval
                or      ax,ax
                jz      IOWAIT9             ; No timeout if zero
                mov     ds:[bp],ax          ; Timeout counter set to timeout interval
                mov     byte ptr ds:13[si],200; Timeout enable (1 second)
IOWAIT9:
                pop     ds
                pop     bp
                pop     di
                pop     si
                pop     bx
                pop     ax
                ret
IOWAIT          endp
```

ISOPEN.ASM

The ISOPEN utility in this include file checks if a channel is open. The primary use of ISOPEN is for configuration programs to determine if a handler is already open.

When a handler is closed, configuration programs create a parameter scratch area, write configuration parameters into it, and put the scratch area address in the appropriate entry in the handler information table. If the scratch area already exists, the handler information table entry will already point to the scratch area, and the configuration program will write its parameters into the existing scratch area.

When a handler is open, however, the entry in the handler information table is the address of the handler scratch area, not of the parameter scratch area. If the configuration program is run after the handler is open, it could misinterpret the handler information table entry, and modify the handler scratch area by mistake. ISOPEN allows configuration programs to check if the handler is open regardless of the meaning of the entry in the handler information table. This allows configuration programs to take different action depending on whether or not the handler is open. Refer to the "User-Defined Handlers" chapter in part 1, "Operating System", for information about using the handler information table.

Program listing:

```
                        .sfcond
                        if1
;*************************************************************
;*     "(c) Copyright Hewlett-Packard Company, 1986.   All   *
;*     rights are reserved.  Copying or other reproduction   *
;*     of this program for inclusion in an application or    *
;*     for archival purposes is permitted without the prior  *
;*     written consent of Hewlett-Packard Company."          *
;*************************************************************
                        endif
                        .lfcond

;*********************************************************
;*
;* Name: ISOPEN
;*
;* Version: 1.3
;*
;* Description:
;*   Check if a channel is open
;*
;* Call with:
;*   AL = Channel number
;*
;* Returns:
;*   AL = Error code:
;*        00h       No error (channel open)
;*        65h (101) Illegal parameter
;*        69h (105) Channel not open
;*
;* Registers altered:
;*   AL
;*
;* Notes:
;*   None
;*
```

```
;*****************************************************

OBIT            EQU     10h

ISOPEN          proc    near
                push    ds
                push    si
                push    ax
                mov     si,16h
                mov     ds,si
                mov     si,ds:[0Ah]         ; Size of open table
                sub     ah,ah               ; Set AH=0
                cmp     ax,si               ; Check for valid channel #
                jae     ISOPEN_ERR
                mov     si,ds:[24h]         ; Open channel table
                mov     ds,ds:[00h]         ; DS = operating system data segment
                mov     ah,0Ch              ; 0Ch (12) bytes per open table entry
                mul     ah                  ; Result to AX
                add     si,ax               ; DS:SI points to channel entry
                pop     ax                  ; Restore AH
                mov     al,69h              ; Preload "Channel not open"
                test    byte ptr ds:[si],OBIT; Is channel open?
                jz      ISOPEN1             ; Channel not open.
                xor     al,al               ; Channel is open.  Return 00h.
ISOPEN1:
                pop     si
                pop     ds
                ret
;-
ISOPEN_ERR:
                pop     ax                  ; Restore AH
                mov     al,65h              ; Illegal parameter
                jmp     ISOPEN1

ISOPEN          endp
```

LLHLINKG.ASM

The handler linkage routines are in LLHLINKG.ASM. This set of utilities is a single include file that can be used as part of a high-level handler to call a low-level handler. Below is a list of all the linkage routines.

Table M-5. Handler Linkage Routine List

Name	Description
LLH_CLOSE	Call CLOSE routine of low-level handler
LLH_IOCTL	Call IOCTL routine of low-level handler
LLH_OPEN	Call OPEN routine of low-level handler
LLH_READ	Call READ routine of low-level handler
LLH_RSVD2	Call RSVD2 routine of low-level handler
LLH_RSVD3	Call RSVD3 routine of low-level handler
LLH_TERM	Call TERM routine of low-level handler
LLH_WARM	Call WARM routine of low-level handler
LLH_WRITE	Call WRITE routine of low-level handler

All the information about the linkage routines and how to use them is in the "User-Defined Handlers" chapter in part 1, "Operating System".

Program listing:

```
                        .sfcond
                        if1
;*************************************************************
;*     "(c) Copyright Hewlett-Packard Company, 1986.   All   *
;*     rights are reserved.  Copying or other reproduction   *
;*     of this program for inclusion in an application or    *
;*     for archival purposes is permitted without the prior  *
;*     written consent of Hewlett-Packard Company."          *
;*************************************************************
                        endif
                        .lfcond

; List of functions:
;
; LLH_OPEN  - calls OPEN routine of specified handler
;
; LLH_CLOSE  - calls CLOSE routine of specified handler
;
; LLH_READ  - calls READ routine of specified handler
;
; LLH_WRITE  - calls WRITE routine of specified handler
;
; LLH_WARM  - calls WARM routine of specified handler
;
; LLH_TERM  - calls TERM routine of specified handler
;
; LLH_IOCTL  - calls IOCTL routine of specified handler
;
; LLH_RSVD2  - calls RSVD2 routine of specified handler
;
; LLH_RSVD3  - calls RSVD3 routine of specified handler
```

```
;
                        page
AXREG                   equ         00h
ALREG                   equ         00h
AHREG                   equ         01h
;
BXREG                   equ         02h
BLREG                   equ         02h
BHREG                   equ         03h
;
CXREG                   equ         04h
CLREG                   equ         04h
CHREG                   equ         05h
;
DXREG                   equ         06h
DLREG                   equ         06h
DHREG                   equ         07h
;
SIREG                   equ         08h
DIREG                   equ         0Ah
DSREG                   equ         0Ch
ESREG                   equ         0Eh
BPREG                   equ         10h

OBIT                    equ         10h

save_regs               macro
;
; Save registers on stack in the order used by handler call
;
                        push        ds
                        push        bp
                        pushf
                        push        cs
                        mov         bp,offset RET_TO_HLH
                        push        bp

                        push        bp
                        push        es
                        push        ds
                        push        di
                        push        si
                        push        dx
                        push        cx
                        push        bx
                        push        ax
                        mov         bp,sp
                        mov         ss:BPREG[bp],bp

                        endm


restore_regs            macro
;
; Pop registers off of stack and return to high-level handler
;
                        pop         bx                  ; Get BH into AH
                        mov         ah,bh
                        pop         bx
                        pop         cx
```

```
                        pop     dx
                        pop     si
                        pop     di
                        pop     ds
                        pop     es
                        pop     bp
                        iret
                        endm
                        page
;*****************************************************
;*
;* Name: LLH_OPEN
;*
;* Version: 1.5
;*
;* Description:
;*   Call the OPEN routine of the low-level handler
;*
;* Call with:
;*   AL = Channel number to open
;*   ES:BX = Address of handler name string to open
;*
;* Returns:
;*   AL = Error code:
;*        00h       No errors
;*        65h (101) Illegal parameter
;*        66h (102) Directory does not exist
;*        67h (103) File not found
;*        6Ah (106) Channel already open
;*        6Eh (110) Access restricted
;*        (any others returned by handler)
;*
;* Registers altered:
;*   AL
;*
;* Notes:
;*   None.
;*
;*****************************************************

LLH_OPEN                proc    near

                        save_regs                       ; Save regs on stack
;
; Save DI and SI for FIND_HNDLR
;
                        mov     di,es
                        mov     si,bx
;
; Get segment addr of work area out of channel table
;
                        cmp     al,5                    ; Valid device channel
                        jl      LLH_OPEN_1              ; Yes, continue
                        mov     al,65h                  ; Invalid parameter
                        jmp     short LLH_OPEN_5        ; Exit with return code in AL

LLH_OPEN_1:
                        mov     bx,16h                  ; Table of addresses
                        mov     ds,bx
                        mov     bx,ds:24h               ; Get offset of open table
```

```
                        mov     ds,ds:00h
                        mov     ah,0Ch              ; Length of each entry in open table
                        mul     ah                  ; Multiply by channel number
                        add     bx,ax
                        test    byte ptr ds:[bx],08IT; Is channel open?
                        jnz     LLH_OPEN_2          ; Yes, continue
                        mov     al,69h              ; Channel not open
                        jmp     short LLH_OPEN_5    ; Exit with return code in AL
LLH_OPEN_2:
                        mov     cx,ds:5[bx]         ; Get stored value in open table
                        push    ds                  ; Address and value of
                        push    bx                  ; DS entry in open table
                        push    cx
;
; Set up my return address for open entry of handler
;
                        push    cs
                        mov     dx,offset LLH_OPEN_4 ; Return address this program
                        push    dx                  ; Return address in DX
;
; Set up address of handler entry on stack for IRET to branch to
;
                        call    FIND_HNDLR          ; Get segment address of handler
                        and     al,al               ; Any errors
                        je      LLH_OPEN_3          ; No, continue
                        add     sp,4                ; Yes, clean up stack
                        jmp     short LLH_OPEN_4    ; Return with error code in AL
;-
LLH_OPEN_3:
                        mov     ds:08h[bx],cx       ; Save handler CS
                        pushf                       ; Put flags for IRET
                        push    cx                  ; Segment address of handler
                        mov     bx,6                ; Offset of OPEN entry
                        push    bx
;
; Restore registers to the values they had when function was called
;
                        cli
                        add     sp,10h              ; Point to register values
                        pop     ax
                        pop     bx
                        pop     cx
                        pop     dx
                        pop     si
                        pop     di
                        pop     ds
                        pop     es
                        pop     bp
;
; Branch to handler entry
;
                        sub     sp,9*2+10h          ; Point to IRET addr
                        iret
;-
;
; Return to calling program with return code in AL
; NOTE: it is the responsibility of the handler entry that was
; just executed to set AL to appropriate return code
;
                                                    ; Address in DX
```

```
LLH_OPEN_4:
                        pop         cx                  ; Get the saved copy of segment address
                                                        ; of high-level handler's work area
                        pop         bx
                        pop         ds
                        xchg        ds:05h[bx],cx       ; Restore to open table and get
                                                        ; saved copy of segment address of
                                                        ; low-level handler's work area
                        mov         ds:0Ah[bx],cx       ; Save low-level handler's work area seg
LLH_OPEN_5:
                        restore_regs                    ; Return to high-level handler
;-
LLH_OPEN                endp
                        page
;*********************************************************
;*
;* Name: LLH_CLOSE
;*
;* Version: 1.5
;*
;* Description:
;*   Call the CLOSE routine of the low-level handler
;*
;* Call with:
;*   AL = Channel number to close
;*
;* Returns:
;*   AL = Error code
;*
;* Registers altered:
;*   AL
;*
;* Notes:
;*   None.
;*
;*********************************************************


LLH_CLOSE               proc        near

                        save_regs
                        mov         bx,09h              ; Offset of CLOSE entry
                        jmp         CALL_HNDLR          ; Go to handler
LLH_CLOSE               endp
                        page
;*********************************************************
;*
;* Name: LLH_READ
;*
;* Version: 1.5
;*
;* Description:
;*   Call the READ routine of the low-level handler
;*
;* Call with:
;*   AL = Channel number from which to read
;*   CX = Number of bytes to read
;*   ES:BX = Address of read buffer
;*
;* Returns:
```

```
;*   AL = Error code
;*   CX = Number of bytes actually read
;*
;* Registers altered:
;*   AL,CX
;*
;* Notes:
;*   None
;*
;*************************************************************

LLH_READ                proc        near

                        save_regs
                        mov         bx,0Ch              ; Offset of READ entry
                        jmp         CALL_HNDLR          ; Go to handler
LLH_READ                endp
                        page
;*************************************************************
;*
;* Name: LLH_WRITE
;*
;* Version: 1.5
;*
;* Description:
;*   Call the WRITE routine of the low-level handler
;*
;* Call with:
;*   AL = Channel number to write
;*   CX = Number of bytes to write
;*   ES:BX = Address of write buffer
;*
;* Returns:
;*   AL = Error code
;*   CX = Number of bytes actually written
;*
;* Registers altered:
;*   AL,CX
;*
;* Notes:
;*   None
;*
;*************************************************************

LLH_WRITE               proc        near

                        save_regs
                        mov         bx,0Fh              ; Offset of WRITE entry
                        jmp         CALL_HNDLR          ; Go to handler
LLH_WRITE               endp
                        page

;*************************************************************
;*
;* Name: LLH_WARM
;*
;* Version: 1.5
;*
;* Description:
;*   Call the WARM routine of the low-level handler
```

```
;*
;* Call with:
;*   AL = Channel number
;*
;* Returns:
;*   AL = Error code
;*
;* Registers altered:
;*   AL
;*
;* Notes:
;*   None
;*
;*********************************************************

LLH_WARM                proc        near

                        save_regs
                        mov         bx,12h              ; Offset of WARM entry
                        jmp         CALL_HNDLR          ; Go to handler
LLH_WARM                endp

                        page
;*********************************************************
;*
;* Name: LLH_TERM
;*
;* Version: 1.5
;*
;* Description:
;*   Call the TERM routine of the low-level handler
;*
;* Call with:
;*   AL = Channel number
;*   AH = Cause of termination
;*
;* Returns:
;*   AL = Error code
;*
;* Registers altered:
;*   AL
;*
;* Notes:
;*   Note that entry conditions are different for LLH_TERM
;*   than those which are seen by the low-level handler (AH, AL).
;*   LLH_TERM moves AH (cause of termination) into AL before
;*   calling the low-level handler.
;*
;*********************************************************

LLH_TERM                proc        near

                        save_regs
                        pop         bx                  ; Get original AX into BX
                        xchg        bh,bl               ; Exchange AH and AL
                        push        bx                  ; Put back on the stack
                        mov         bx,15h              ; Offset of TERM entry
                        jmp         CALL_HNDLR          ; Go to handler
LLH_TERM                endp
                        page
```

```
;*************************************************************
;*
;* Name: LLH_IOCTL
;*
;* Version: 1.5
;*
;* Description:
;*   Call the IOCTL routine of the low-level handler
;*
;* Call with:
;*   AL = Channel number
;*   AH = IOCTL function code
;*   (others as defined by handler)
;*
;* Returns:
;*   AL = Error code
;*   (others as defined by handler)
;*
;* Registers altered:
;*   AL
;*   (others as defined by handler)
;*
;* Notes:
;*   None
;*
;*************************************************************

LLH_IOCTL               proc        near

                        save_regs
                        mov         bx,1Bh              ; Offset of IOCTL entry
                        jmp         CALL_HNDLR          ; Go to handler
LLH_IOCTL               endp
                        page
;*************************************************************
;*
;* Name: LLH_RSVD2
;*
;* Version: 1.5
;*
;* Description:
;*   Call the RSVD2 routine of the low-level handler
;*
;* Call with:
;*   AL = Channel number
;*   (others as defined by handler)
;*
;* Returns:
;*   AL = Error code
;*   (others as defined by handler)
;*
;* Registers altered:
;*   AL
;*   (others as defined by handler)
;*
;* Notes:
;*   None
;*
;*************************************************************
```

```
LLH_RSVD2               proc        near

                        save_regs
                        mov         bx,1Eh              ; Offset of RSVD2 entry
                        jmp         CALL_HNDLR          ; Go to handler
LLH_RSVD2               endp
                        page
;*************************************************************
;*
;* Name: LLH_RSVD3
;*
;* Version: 1.5
;*
;* Description:
;*   Call the RSVD3 routine of the low-level handler
;*
;* Call with:
;*   AL = Channel number
;*   (others as defined by handler)
;*
;* Returns:
;*   AL = Error code
;*   (others as defined by handler)
;*
;* Registers altered:
;*   AL
;*   (others as defined by handler)
;*
;* Notes:
;*   None
;*
;*************************************************************

LLH_RSVD3               proc        near

                        save_regs
                        mov         bx,21h              ; Offset of RSVD3 entry
                        jmp         CALL_HNDLR          ; Go to handler
LLH_RSVD3               endp

                        page
;*************************************************************
;*
;* Name: CALL_HNDLR
;*
;* Version: 1.5
;*
;* Description:
;*   Call the selected routine of the low-level handler
;*
;* Call with:
;*   AL = Channel number
;*   BX = Offset of handler entry to call
;*
;* Returns:
;*   AL = Error code
;*
;* Registers altered:
;*   AL
;*
```

```
;* Notes:
;*   None
;*
;*****************************************************************

CALL_HNDLR              proc        near

;
; Set up my return address for handler entry to return to
;
                        push        cs
                        mov         dx,offset CALL_HNDLR_EXIT; Return address this program
                        push        dx                  ; Return address in DX
;
; Set up address of handler entry on stack for IRET to branch to
;
                        pushf                           ; Put flags for IRET
                        mov         si,16h
                        mov         ds,si
                        mov         si,ds:24h           ; DS:SI point to open table
                        mov         ds,ds:00h
                        mov         ah,0Ch
                        mul         ah                  ; Multiply channel number by length of each entry
                        add         si,ax               ; DS:SI points to table entry for this channel
                        mov         cx,ds:0Ah[si]       ; Get low-level handler's work area segment
                        mov         ss:DSREG[bp],cx     ; Send to low-level handler
                        mov         cx,ds:08h[si]       ; Get low-level handler's CS
CALL_HNDLR_1:
                        push        cx                  ; Segment address of handler
                        push        bx                  ; Offset of handler entry
;
; Restore registers to the values they had when function was called
;
; Stack now:
; 0Ah                   (saved registers)
; 08h                   My CS
; 06h                   My IP (points to CALL_HNDLR_EXIT)
; 04h                   flags
; 02h                   low-level handler's segment address
; 00h                   low-level handler's offset address
;
                        cli
                        add         sp,0Ah              ; Point to register values
                        pop         ax
                        pop         bx
                        pop         cx
                        pop         dx
                        pop         si
                        pop         di
                        pop         ds
                        pop         es
                        pop         bp
;
; Branch to handler entry
;
                        sub         sp,9*2+0Ah          ; Point to IRET addr
                        iret
;-
;
; Return to calling program with return code in AL
```

```
; NOTE: it is the responsibility of the handler entry that was
; just executed to set AL to appropriate return code
;
CALL_HNDLR_EXIT:
                        restore_regs                    ; Return to high-level handler

CALL_HNDLR              endp
                        page
;*********************************************************
;*
;* Name: FIND_HNDLR
;*
;* Version: 1.5
;*
;* Description:
;*   Find a handler program whose name is at DI:SI
;*
;* Call with:
;*   DI:SI = Pointer to handler name
;*
;* Returns:
;*   AL = Error code
;*        00h       No errors
;*        65h (101) Illegal parameter
;*        66h (102) Directory does not exist
;*        67h (103) File not found
;*        6Eh (110) Access restricted
;*   CX = Segment address of handler
;*
;* Registers altered:
;*   AX,CX
;*
;* Notes:
;*   None
;*
;*********************************************************

FIND_HNDLR              proc        near

                        push        bx
                        push        dx
                        push        di
                        push        bp
                        push        ds
                        push        es
                        mov         bp,sp               ; Save SP
                        sub         sp,0Eh              ; Get room for file information buffer
                        mov         bx,si               ; Set ES:BX to file name address
                        mov         es,di
                        mov         di,sp               ; Get offset address of file information buffer
                        push        ss
                        pop         ds                  ; Get segment address of file information buffer
                        mov         ah,16h              ; FIND_FILE function
FIND_HNDLR1:
                        mov         dx,di               ; Offset address of file information buffer
                        int         1Ah
;
; If there are any errors, exit with appropriate error code
;
                        or          al,al               ; Any errors?
```

```
                        jnz         FIND_HNDLR_EXIT     ; Yes, exit with error code in AL

;
; If this is not a handler program (type "H"), generate "Access restricted"
; Get another file with same name and correct type
;
                        mov         al,6Eh              ; "Access restricted" error code
                        cmp         byte ptr ss:7[di],"H"; Is it a handler?
                        jne         FIND_HNDLR_EXIT     ; No, exit with error
                        xor         al,al               ; Set "No error" code
                        mov         cx,ss:08h[di]       ; Get segment address of handler

FIND_HNDLR_EXIT:
                        mov         sp,bp
                        pop         es
                        pop         ds
                        pop         bp
                        pop         di
                        pop         dx
                        pop         bx
                        ret
FIND_HNDLR              endp


RET_TO_HLH              proc        near
;
; Restore BP and DS, then return to calling program
;
                        pop         bp
                        pop         ds
                        ret
RET_TO_HLH              endp
```

NOIOWAIT.ASM

The NOIOWAIT utility in this include file disables the operating system I/O wait state. This is the state in which low battery, power switch, and timeout errors can be reported by the IOABORT utility while handler READ or WRITE routines are waiting for I/O.

The I/O wait state is enabled with IOWAIT. NOIOWAIT should be called when the handler READ or WRITE routines end, whether the routines are ending because I/O is being aborted, or because of a normal end. Refer to IOWAIT and IOABORT for further information.

Program listing:

```
                        .sfcond
                        if1
;*************************************************************
;*     "(c) Copyright Hewlett-Packard Company, 1986.   All   *
;*     rights are reserved.  Copying or other reproduction   *
;*     of this program for inclusion in an application or    *
;*     for archival purposes is permitted without the prior  *
;*     written consent of Hewlett-Packard Company."          *
;*************************************************************
                        endif
                        .lfcond

;*************************************************************
;*
;* Name: NOIOWAIT
;*
;* Version: 1.3
;*
;* Description:
;*   Disable I/O wait state
;*
;* Call with:
;*   None
;*
;* Returns:
;*   None
;*
;* Registers altered:
;*   None
;*
;* Notes:
;*   None.
;*
;*************************************************************


NOIOWAIT                proc    near
                        push    bx
                        push    si
                        push    ds
                        mov     bx,16h
                        mov     ds,bx
                        mov     bx,ds:[14h]
                        add     bx,4                ; BX points to current status
                        mov     si,ds:[1Ah]         ; System timer event control table
                        mov     ds,ds:[00h]         ; DS = operating system data segment
                        and     byte ptr ds:[bx],0DFh; I/O waiting flag clear
```

```
                mov     byte ptr ds:13[si],-1; Timeout disable
                pop     ds
                pop     si
                pop     bx
                ret
NOIOWAIT        endp
```

READCTRL.ASM

The READCTRL utility in this include file reads the saved copy of the main control register (I/O address 0Bh). When hardware status is changed by an assembly language program, the program must use the following procedure to ensure that hardware devices unaffected by the change remain in their current state:

- Read the saved copy of the main control register using READCTRL.
- Change the bits needed to cause the hardware status to change.
- Write the updated value back to its saved location and output the updated value to the main control register using SETCTRL.

Refer to the "Hardware Control and Status Registers" chapter in part 1, "Operating System", for further information.

Program listing:

```
                        .sfcond
                        if1
;*************************************************************
;*     "(c) Copyright Hewlett-Packard Company, 1986.   All   *
;*     rights are reserved.  Copying or other reproduction   *
;*     of this program for inclusion in an application or    *
;*     for archival purposes is permitted without the prior  *
;*     written consent of Hewlett-Packard Company."          *
;*************************************************************
                        endif
                        .ifcond

;*************************************************************
;*
;* Name: READCTRL
;*
;* Version: 1.3
;*
;* Description:
;*   Read the saved copy of the main control register
;*   (I/O address 0Bh)
;*
;* Call with:
;*   None
;*
;* Returns:
;*   AH = Main control register value
;*
;* Registers altered:
;*   AH
;*
;* Notes:
;*   The control register bits have the following meanings:
;*
;*  ...............................................
;*  .     .     .     .     .     .     .     .     .
;*  . 7   . 6   . 5   . 4   . 3   . 2   . 1   . 0   .
;*  .     .     .     .     .     .     .     .     .
;*  ...............................................
;*                      .     .     .     .     .
;*                      .     .     .     .     .
```

```
;*                          .    .    .    .    .
;*                          .    .    .    .......
;*                          .    .    .       .
;*                          .    .    .       .
;*                          .    .    .       ... beeper control
;*                          .    .    .           11 = low tone
;*                          .    .    .           01 = high tone
;*                          .    .    .           00 = off
;*                          .    .    .
;*                          .    .    .
;*                          .    .    .... serial port power control
;*                          .    .         1 = enable
;*                          .    .         0 = disable
;*                          .    .
;*                          .    .....bar code port power control
;*                          .         1 = enable
;*                          .         0 = disable
;*                          .
;*                          .....bar code port transition control
;*                               1 = enable
;*                               0 = disable
;*
;*****************************************************

READCTRL                proc        near

                        push        bx
                        push        ds
                        mov         bx,16h              ; Get operating system pointer table
                        mov         ds,bx
                        mov         bx,ds:[14h]         ; Get offset of status area
                        mov         ds,ds:[00h]
                        inc         bx
                        inc         bx                  ; Point to saved copy of main control register
                        mov         ah,ds:[bx]          ; Read it
                        pop         ds
                        pop         bx
                        ret

READCTRL                endp
```

READINTR.ASM

The READINTR utility in this include file reads the saved copy of the interrupt control register (I/O address 00h). When interrupt status is changed by an assembly language program, the program must use the following procedure to ensure that interrupts unaffected by the change remain in their current state:

- Read the saved copy of the interrupt control register using READINTR.
- Change the bits needed to cause the interrupt status to change.
- Write the updated value back to its saved location and output the updated value to the interrupt control register using SETINTR.

Refer to the "Hardware Control and Status Registers" and "Interrupt Controller" chapters in part 1, "Operating System", for further information.

Program listing:

```
                        .sfcond
                        if1
;*************************************************************
;*     "(c) Copyright Hewlett-Packard Company, 1986.   All   *
;*     rights are reserved.  Copying or other reproduction   *
;*     of this program for inclusion in an application or    *
;*     for archival purposes is permitted without the prior  *
;*     written consent of Hewlett-Packard Company."          *
;*************************************************************
                        endif
                        .lfcond

;*************************************************************
;*
;* Name: READINTR
;*
;* Version: 1.3
;*
;* Description:
;*   Read the saved copy of the interrupt control register
;*   (I/O address 00h)
;*
;* Call with:
;*   None
;*
;* Returns:
;*   AH = interrupt control register value
;*
;* Registers altered:
;*   AH
;*
;* Notes:
;*   The interrupt control register bits have the following meanings:
;*
;*     (bit set to 1 -> corresponding interrupt enabled)
;*     (bit set to 0 -> corresponding interrupt disabled)
;*
;*   ...............................................
;*   .     .     .     .     .     .     .     .     .
;*   . 7   . 6   . 5   . 4   . 3   . 2   . 1   . 0   .
;*   .     .     .     .     .     .     .     .     .
```

```
;*   ...............................................
;*      .    .    .    .    .    .    .    .
;*      .    .    .    .    .    .    .    .
;*      .    .    .    .    .    .    .    .
;*      .    .    .    .    .    .    .    ... system timer interrupt
;*      .    .    .    .    .    .    .        (type 50h)
;*      .    .    .    .    .    .    .
;*      .    .    .    .    .    .    ... bar code timer interrupt
;*      .    .    .    .    .    .        (type 51h)
;*      .    .    .    .    .    .    .
;*      .    .    .    .    .    .    ... bar code port transition interrupt
;*      .    .    .    .    .    .        (type 52h)
;*      .    .    .    .    .    .
;*      .    .    .    .    .    ... serial port data received interrupt
;*      .    .    .    .    .        (type 53h)
;*      .    .    .    .    .
;*      .    .    .    .    ... low main battery voltage interrupt
;*      .    .    .    .        (type 54h)
;*      .    .    .    .
;*      .    .    .    ... power switch interrupt
;*      .    .    .        (type 55h)
;*      .    .    .
;*      .    .    ... reserved interrupt 1
;*      .    .        (type 56h)
;*      .    .
;*      .    ... reserved interrupt 2
;*           (type 57h)
;*
;*********************************************************

READINTR            proc        near

                    push        bx
                    push        ds
                    mov         bx,16h              ; Get operating system pointer table
                    mov         ds,bx
                    mov         bx,ds:[14h]         ; Get offset of status area
                    mov         ds,ds:[00h]
                    add         bx,3                ; Point to saved copy of interrupt control register
                    mov         ah,ds:[bx]          ; Read it
                    pop         ds
                    pop         bx
                    ret

READINTR            endp
```

SCANKYBD.ASM

The SCANKYBD utility in this include file scans the keyboard and returns the keycode of the first key found down. Normally, the system timer interrupt service routine causes the keyboard to be scanned every 5 ms. However, in time-critical handlers such as for the bar code port, the system timer may be disabled while waiting for data to be received at the port (to prevent bar code port transition interrupts from being missed). SCANKYBD performs the operating system keyboard scan operation when the system timer is disabled.

The SCANKYBD utility can be called while the handler waits for data, thereby allowing the handler to respond to keyboard input. This helps prevent users from gaining the perception of no machine activity that accompanies no keyboard response.

The keyboard columns are scanned from right to left, and the rows from top to bottom. The first key found down in that scanning sequence will be reported as a keycode. Other keys to the left or below the first key found will be ignored.

The keycodes corresponding to each key position and the corresponding ASCII characters are described in the "Keyboard" chapter in part 1, "Operating System".

SCANKYBD uses FINDOS to find the operating system file and the start of the operating system jump table.

Program listing:

```
                        .sfcond
                        if1
;*************************************************************
;*     "(c) Copyright Hewlett-Packard Company, 1987.   All   *
;*     rights are reserved.  Copying or other reproduction   *
;*     of this program for inclusion in an application or    *
;*     for archival purposes is permitted without the prior  *
;*     written consent of Hewlett-Packard Company."          *
;*************************************************************
                        endif
                        .lfcond

                        include findos.asm

;*************************************************************
;*
;* Name: SCANKYBD
;*
;* Version: 1.3
;*
;* Description:
;*   Call the operating system routine to scan the keyboard
;*
;* Call with:
;*   None
;*
;* Returns:
;*   AL = 0 if no key down
;*   AL = HP-94 keycode (see chapter 8, "Keyboard", for keycode values)
;*
;* Registers altered:
```

```
;*   AX
;*
;* Notes:
;*   The key is NOT debounced by this routine - the routine
;*   simply reports what key is down at the present time.
;*
;*****************************************************************

SCANKYBD                proc        near
                        push        bx
                        push        si
                        push        ds
                        push        es
                        call        FINDOS
; DS is SYOS segment
; ES is operating system pointer table segment
                        push        ds
                        push        es:[36h]            ; Get offset of keyscan routine
                        mov         si,es:[00h]
                        mov         ds,si               ; DS = operating system data segment
                        mov         si,sp
                        call        dword ptr ss:[si]
                        add         sp,4
                        pop         es
                        pop         ds
                        pop         si
                        pop         bx
                        ret
SCANKYBD                endp
```

SETCTRL.ASM

The SETCTRL utility in this include file writes a value to the location of the saved copy of the main control register (I/O address 0Bh), then writes the value to the main control register as well. When hardware status is changed by an assembly language program, the program must use the following procedure to ensure that hardware devices unaffected by the change remain in their current state:

- Read the saved copy of the main control register using READCTRL.
- Change the bits needed to cause the hardware status to change.
- Write the updated value back to its saved location and output the updated value to the main control register using SETCTRL.

Refer to the "Hardware Control and Status Registers" chapter in part 1, "Operating System", for further information.

Program listing:

```
                        .sfcond
                        if1
;*************************************************************
;*     "(c) Copyright Hewlett-Packard Company, 1986.   All   *
;*     rights are reserved.  Copying or other reproduction   *
;*     of this program for inclusion in an application or    *
;*     for archival purposes is permitted without the prior  *
;*     written consent of Hewlett-Packard Company."          *
;*************************************************************
                        endif
                        .lfcond

;*************************************************************
;*
;* Name: SETCTRL
;*
;* Version: 1.3
;*
;* Description:
;*   Set the mail control register (I/O address 0Bh) and its
;*   saved copy to the value specified in AH
;*
;* Call with:
;*   AH = main control register value
;*
;* Returns:
;*   The main control register and its saved copy are set
;*   to the value in AH
;*
;* Registers altered:
;*   None
;*
;* Notes:
;*   The control register bits have the following meanings:
;*
;*  ...............................................
;*  .    .    .    .    .    .    .    .    .
;*  . 7  . 6  . 5  . 4  . 3  . 2  . 1  . 0  .
;*  .    .    .    .    .    .    .    .    .
;*  ...............................................
;*                       .    .    .    .    .
```

```
;*                    .    .    .    .    .
;*                    .    .    .    .    .
;*                    .    .    .    .......
;*                    .    .    .       .
;*                    .    .    .       .
;*                    .    .    .       ... beeper control
;*                    .    .    .           11 = low tone
;*                    .    .    .           01 = high tone
;*                    .    .    .           00 = off
;*                    .    .    .
;*                    .    .    .... serial port power control
;*                    .    .         1 = enable
;*                    .    .         0 = disable
;*                    .    .
;*                    .    .....bar code port power control
;*                    .         1 = enable
;*                    .         0 = disable
;*                    .
;*                    .....bar code port transition control
;*                         1 = enable
;*                         0 = disable
;*
;*****************************************************

SETCTRL             proc        near

                    push        ax
                    push        bx
                    push        ds
                    pushf                           ; Save interrupt flag (CLI below)
                    mov         bx,16h              ; Get operating system pointer table
                    mov         ds,bx
                    mov         bx,ds:[14h]         ; Get offset of status area
                    mov         ds,ds:[00h]
                    inc         bx
                    inc         bx                  ; Point to saved copy of main control register
                    mov         al,ah               ; Get AH into AL
                    cli
                    mov         ds:[bx],al          ; Send value to saved copy location
                    out         08h,al              ; Send to main control register
                    popf                            ; Restore interrupt flag
                    pop         ds
                    pop         bx
                    pop         ax
                    ret

SETCTRL             endp
```

SETINTR.ASM

The SETINTR utility in this include file writes a value to the location of the saved copy of the interrupt control register (I/O address 00h), then writes the value to the interrupt control register as well. When interrupt status is changed by an assembly language program, the program must use the following procedure to ensure that interrupts unaffected by the change remain in their current state:

- Read the saved copy of the interrupt control register using READINTR.
- Change the bits needed to cause the interrupt status to change.
- Write the updated value back to its saved location and output the updated value to the interrupt control register using SETINTR.

Refer to the "Hardware Control and Status Registers" and "Interrupt Controller" chapters in part 1, "Operating System", for further information.

Program listing:

```
                        .sfcond
                        if1
;*************************************************************
;*     "(c) Copyright Hewlett-Packard Company, 1986.   All   *
;*     rights are reserved.  Copying or other reproduction   *
;*     of this program for inclusion in an application or    *
;*     for archival purposes is permitted without the prior  *
;*     written consent of Hewlett-Packard Company."          *
;*************************************************************
                        endif
                        .lfcond

;*************************************************************
;*
;* Name: SETINTR
;*
;* Version: 1.3
;*
;* Description:
;*   Set the interrupt control register (I/O address 00h)
;*   and its saved copy to the value in AH
;*
;* Call with:
;*   AH = interrupt control register value
;*
;* Returns:
;*   The interrupt control register and its saved copy are
;*   set to the value in AH
;*
;* Registers altered:
;*   None
;*
;* Notes:
;*   The interrupt control register bits have the following meanings:
;*
;*     (bit set to 1 -> corresponding interrupt enabled)
;*     (bit set to 0 -> corresponding interrupt disabled)
;*
;*   ...............................................
;*   .   .   .   .   .   .   .   .   .   .   .   .
;*   . 7 . 6 . 5 . 4 . 3 . 2 . 1 . 0 .
```

```
;*   .    .    .    .    .    .    .    .    .
;*   ..............................................
;*      .    .    .    .    .    .    .    .
;*      .    .    .    .    .    .    .    .
;*      .    .    .    .    .    .    .    .
;*      .    .    .    .    .    .    .    ... system timer interrupt
;*      .    .    .    .    .    .    .        (type 50h)
;*      .    .    .    .    .    .    .
;*      .    .    .    .    .    .    ... bar code timer interrupt
;*      .    .    .    .    .    .        (type 51h)
;*      .    .    .    .    .    .
;*      .    .    .    .    .    ... bar code port transition interrupt
;*      .    .    .    .    .        (type 52h)
;*      .    .    .    .    .
;*      .    .    .    .    ... serial port data received interrupt
;*      .    .    .    .        (type 53h)
;*      .    .    .    .
;*      .    .    .    ... low main battery voltage interrupt
;*      .    .    .        (type 54h)
;*      .    .    .
;*      .    .    ... power switch interrupt
;*      .    .        (type 55h)
;*      .    .
;*      .    ... reserved interrupt 1
;*      .        (type 56h)
;*      .
;*      ... reserved interrupt 2
;*          (type 57h)
;*
;*************************************************************

SETINTR                 proc near

                        push    ax
                        push    bx
                        push    ds
                        pushf                           ; Save interrupt flag (CLI below)
                        mov     bx,16h                  ; Get operating system pointer table
                        mov     ds,bx
                        mov     bx,ds:[14h]             ; Get offset of status area
                        mov     ds,ds:[00h]
                        add     bx,3                    ; Point to saved copy of interrupt control register
                        mov     al,ah                   ; Get AH into AL
                        cli
                        mov     ds:[bx],al              ; Write value to saved copy location
                        out     00h,al                  ; Write to interrupt control register
                        popf                            ; Restore interrupt flag
                        pop     ds
                        pop     bx
                        pop     ax
                        ret

SETINTR                 endp
```

VERSION.ASM

The VERSION utility in this include file returns the version number from the specified program file (type A, B, or H). The version number is the two-byte value at offset 04h in the file — the third pair of bytes in the program header. If the specified program is a handler (type H), VERSION also returns the handler identifier, which is the two-byte identifier at offset 02h in the file — the second pair of bytes in the handler header. VERSION can return the version of the system ROM instead of the version of a program. The system ROM version is part of the copyright message that appears when the machine enters command mode.

Program listing:

```
                        .sfcond
                        if1
;*************************************************************
;*      "(c) Copyright Hewlett-Packard Company, 1987.   All  *
;*      rights are reserved.  Copying or other reproduction  *
;*      of this program for inclusion in an application or   *
;*      for archival purposes is permitted without the prior *
;*      written consent of Hewlett-Packard Company."         *
;*************************************************************
                        endif
                        .lfcond

AXREG                   equ         00h
ALREG                   equ         00h
AHREG                   equ         01h
;
BXREG                   equ         02h
BLREG                   equ         02h
BHREG                   equ         03h
;
CXREG                   equ         04h
CLREG                   equ         04h
CHREG                   equ         05h
;
DXREG                   equ         06h
DLREG                   equ         06h
DHREG                   equ         07h
;
SIREG                   equ         08h
DIREG                   equ         0Ah
DSREG                   equ         0Ch
ESREG                   equ         0Eh
BPREG                   equ         10h

FIND_FILE               equ         16h
BUFFER_SIZE             equ         0Eh

pushregs                macro
                        pushf
                        push        bp
                        push        es
                        push        ds
                        push        di
                        push        si
                        push        dx
                        push        cx
                        push        bx
```

```
                        push        ax
                        mov         bp,sp
                        endm

popregs                 macro
                        pop         ax
                        pop         bx
                        pop         cx
                        pop         dx
                        pop         si
                        pop         di
                        pop         ds
                        pop         es
                        pop         bp
                        popf
                        endm

                        include findos.asm

;*************************************************************
;*
;* Name: VERSION
;*
;* Version: 1.5
;*
;* Description:
;*   Return the version number of the specified file
;*
;* Call with:
;*   ES:BX points to a file name (see Notes, below)
;*
;* Returns:
;*   AL = Error code:
;*        00h       No error
;*        65h (101) Illegal parameter
;*        66h (102) Invalid directory number
;*        67h (103) File not found
;*
;*   If AL=00h:
;*     DX = version (DH = integer part, DL = fractional part)
;*     For type "H" files only:
;*       CX = handler identifier (bytes 2 and 3 of the program header)
;*
;* Registers altered:
;*   CX,DX
;*
;* Notes:
;*   If ES and BX are both zero, the version returned is that
;*   of the system ROM (the version shown in the copyright
;*   message which is displayed when the HP-94 enters command mode).
;*
;*************************************************************

VERSION                 proc        near
                        pushregs
                        mov         ax,es
                        add         ax,bx               ; Check for zero
                        ja          VERSION_FILE
; This is for the system ROM
                        call        FINDOS
```

```
                        mov         ax,ds
                        or          ax,ax
                        mov         al,67h              ; File not found
                        jz          VERSION_RET
; DS is start of SYOS file, ES is start of operating system pointer table
                        mov         si,es:[3Ch]         ; Pointer to version
                        sub         dx,dx               ; Clear out counter
                        mov         bx,010Ah            ; Decade value and "." flag
VERSION_OS1:
                        lodsb
                        sub         al,'9'+1
                        add         al,'9'+1-'0'
                        jnc         VERSION_OS2         ; Not in 0...9
                        xchg        al,dh
                        mul         bl
                        add         dh,al               ; New sum in DH
                        jmp         VERSION_OS1
;-
VERSION_OS2:
                        cmp         al,'.'-'0'
                        jne         VERSION_OS3         ; "." (continue with fraction)
                        sub         bh,01h              ; decrement "." flag
                        jc          VERSION_EXIT        ; already had "." (exit now)
                        xchg        dh,dl
                        jmp         VERSION_OS1
;-
VERSION_OS3:
                        or          bh,bh
                        jnz         VERSION_EXIT
                        xchg        dh,dl
                        jmp         short VERSION_EXIT  ; Done
;-
VERSION_FILE:
                        sub         sp,BUFFER_SIZE      ; allocate file information buffer
                        push        ss
                        pop         ds
                        mov         dx,sp
                        mov         ah,FIND_FILE
                        int         1Ah                 ; O.S. function call
                        or          al,al
                        jnz         VERSION_RET
                        mov         si,sp
                        mov         ds,ss:[si+08h]      ; start segment address of the file
                        mov         al,ss:[si+07h]      ; get file type
                        mov         dx,ds:[04h]         ; Fetch version from file
                        cmp         al,'H'
                        jne         VERSION_EXIT
                        mov         cx,ds:[02h]         ; Fetch handler identifier
                        mov         CXREG[bp],cx
VERSION_EXIT:
                        mov         DXREG[bp],dx
                        sub         al,al               ; result code = 00h
VERSION_RET:
                        mov         ALREG[bp],al        ; return result code to user
                        mov         sp,bp               ; restore stack pointer
                        popregs
                        ret
VERSION                 endp
```

The XIOCTL utility in this include file allows an assembly language program to call the IOCTL routine in an open low-level handler. IOCTL routines are usually called by a high-level handler to cause its low-level handler to take some action, such as change operating configuration or flush its receive buffer. The XIOCTL utility allows any application to direct the behavior of a low-level handler in the same manner. The behavior of the IOCTL routine is described in the "User-Defined Handlers" chapter in part 1, "Operating System", and the "Hewlett-Packard Bar Code Handlers" appendix.

Program listing:

```
                        .sfcond
                        if1
;*************************************************************
;*     "(c) Copyright Hewlett-Packard Company, 1986.   All   *
;*     rights are reserved.  Copying or other reproduction   *
;*     of this program for inclusion in an application or    *
;*     for archival purposes is permitted without the prior  *
;*     written consent of Hewlett-Packard Company."          *
;*************************************************************
                        endif
                        .lfcond

;*************************************************************
;*
;* Name: XIOCTL
;*
;* Version: 1.5
;*
;* Description:
;*   Call the IOCTL routine in a handler
;*
;* Call with:
;*   AL = Channel number
;*   AH = IOCTL function code
;*   Other registers as defined by the handler's IOCTL routine
;*
;* Returns:
;*   AL = Error code:
;*        00h       No error
;*        69h (105) Channel not open
;*   Other registers as defined by the handler's IOCTL routine
;*
;* Registers altered:
;*   AL (handler's AL if channel is open)
;*   Other registers as defined by the handler's IOCTL routine
;*
;* Notes:
;*   See the Technical Reference Manual, Part 1, chapter 3 "User-
;*   Defined Handlers" for more information about IOCTL.
;*
;*   When the IOCTL routine of the handler is called, XIOCTL also
;*   sets up these registers:
;*     DS = Segment address of handler scratch area
;*     BP = Stack offset address of register save area
;*
;*************************************************************
OBIT                    equ         10h
HBIT                    equ         20h
```

```
XIOCTL          proc    near
                pushf
                push    bp
                push    es
                push    ds
                push    di
                push    si
                push    dx
                push    cx
                push    bx
                push    ax
                mov     bp,sp
                mov     bx,16h
                mov     ds,bx
                mov     bx,ds:[0Ah]         ; Size of open table
                cmp     al,bl               ; Check for valid channel #
                jae     XIOCTL_ERROR
                mov     bx,ds:[24h]         ; Open channel table
                mov     ds,ds:[00h]         ; Operating system segment
                mov     ah,0Ch              ; 0Ch (12) bytes per open table entry
                mul     ah                  ; Result to AX
                add     bx,ax               ; DS:BX points to channel entry
                mov     ax,0[bp]            ; Restore AX
                mov     al,byte ptr ds:[bx] ; Read channel status
                and     al,OBIT+HBIT
                cmp     al,OBIT+HBIT        ; Is channel open for a device handler?
                mov     al,69h              ; Preload "Channel not open"
                jne     XIOCTL_END
                xor     al,al
                push    cs
                mov     cx,offset XIOCTL_END
                push    cx                  ; Stack has return address for XIOCTL_END
                push    ds:[bx+3]
                mov     cx,ds:[bx+1]
                add     cx,15h
                push    cx                  ; Stack has execute address for IOCTL
                mov     ds,ds:[bx+5]        ; Load handler data segment
                mov     cx,4[bp]
                mov     bx,2[bp]            ; Restore these registers
                db      0CBh                ; FAR RET (go to IOCTL handler)
;-
XIOCTL_ERROR:
                mov     al,65h              ; Illegal parameter
XIOCTL_END:
                pop     bx                  ; (really AX contents)
                mov     ah,bh               ; Leave AL unchanged from handler IOCTL
                pop     bx
                pop     cx
                pop     dx
                pop     si
                pop     di
                pop     ds
                pop     es
                pop     bp
                popf
                ret
XIOCTL          endp
```

The XTIMEOUT utility in this include file executes a user-defined timeout interrupt routine if one was defined, or turns the machine off if none was defined. It is used by a handler READ or WRITE routine that is waiting for I/O when the IOABORT utility indicates that the timeout occurred. Refer to IOWAIT and IOABORT for further information.

XTIMEOUT uses FINDOS to find the operating system file and the start of the operating system jump table.

Program listing:

```
                        .sfcond
                        if1
;*************************************************************
;*     "(c) Copyright Hewlett-Packard Company, 1987.   All   *
;*     rights are reserved.  Copying or other reproduction   *
;*     of this program for inclusion in an application or    *
;*     for archival purposes is permitted without the prior  *
;*     written consent of Hewlett-Packard Company."          *
;*************************************************************
                        endif
                        .lfcond

                        include findos.asm

;*************************************************************
;*
;* Name: XTIMEOUT
;*
;* Version: 1.3
;*
;* Description:
;*   Execute a user-defined timeout routine, if any.  If no
;*   user-defined timeout routine is present, turn the HP-94
;*   off (will cold start when next turned on)
;*
;* Call with:
;*   None
;*
;* Returns:
;*   None
;*
;* Registers altered:
;*   None
;*
;* Notes:
;*   If there is no user-specified timeout routine, XTIMEOUT
;*   does not return to the caller.  The HP-94 is turned off,
;*   and will cold start when it is turned on again.
;*
;*   If there is a user-specified timeout routine, it will be
;*   executed before XTIMEOUT returns.
;*
;*************************************************************


XTIMEOUT                proc        near
                        push        ax
```

```
                        push    bx
                        push    si
                        push    ds
                        push    es
                        call    FINDOS
; DS is SYOS segment
; ES is operating system pointer table segment
                        push    ds
                        push    es:[3Ah]        ; Get offset of timeout routine
                        mov     si,es:[00h]
                        mov     ds,si           ; Set up operating system data segment
                        mov     si,sp
                        call    dword ptr ss:[si]
                        add     sp,4
                        pop     es
                        pop     ds
                        pop     si
                        pop     bx
                        pop     ax
                        ret
XTIMEOUT                endp
```