Cat. No. W186-E1-4

SYSMAC

C500-ASC04

ASCII Unit

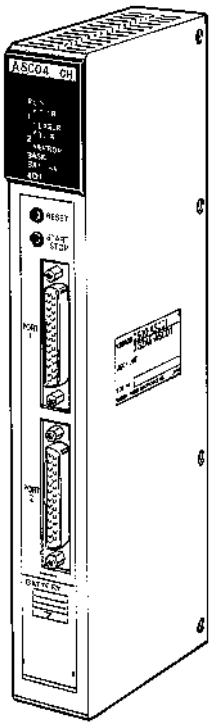
OPERATION MANUAL

OMRON

C500-ASC04 ASCII Unit

Operation Manual

Revised February 2001



Notice:

OMRON products are manufactured for use according to proper procedures by a qualified operator and only for the purposes described in this manual.

The following conventions are used to indicate and classify warnings in this manual. Always heed the information provided with them.

⚠ DANGER Indicates information that, if not heeded, is likely to result in loss of life or serious injury.

⚠ WARNING Indicates information that, if not heeded, could possibly result in loss of life or serious injury.

⚠ Caution Indicates information that, if not heeded, could result in relatively serious or minor injury, damage to the product, or faulty operation.

OMRON Product References

All OMRON products are capitalized in this manual. The word “Unit” is also capitalized when it refers to an OMRON product, regardless of whether or not it appears in the proper name of the product.

The abbreviation “Ch,” which appears in some displays and on some OMRON products, often means “word” and is abbreviated “Wd” in documentation in this sense.

The abbreviation “PC” means Programmable Controller and is not used as an abbreviation for anything else.

Visual Aids

The following headings appear in the left column of the manual to help you locate different types of information.

Note Indicates information of particular interest for efficient and convenient operation of the product.

1, 2, 3... 1. Indicates lists of one sort or another, such as procedures, checklists, etc.

About this Manual:

This manual describes the installation and operation of the C500-ASC04 ASCII Unit. The ASCII Unit can be mounted to a C500, C1000H, C2000H, or CV-series PC to control ASCII data I/O through a BASIC program stored in the ASCII Unit. The C500-ASC04 must be used with a PC that supports the I/O READ and I/O WRITE instructions (READ(88) and WRIT(87) or READ(190) and WRIT(191)).

It has been assumed in the writing of this manual that the reader is already familiar with the hardware, programming, and terminology of OMRON PC's. If a review of this information is necessary, the reader should refer to the appropriate OMRON PC manuals for assistance.

This manual contains the following sections. Please read this manual completely and be sure you understand the information provide before attempting to install and operation the ASCII Unit**.**

Section 1 explains the external hardware of the ASCII Unit and how it connects to a PC system.

Section 2 explains the format of the PC memory area accessed by the ASCII Unit. This area is where the ASCII Unit and the PC exchange data.

Section 3 explains how the ASCII Unit program and the PC program communicate as well as how to write, load, save, and run an BASIC program for the ASCII Unit.

Section 4 presents the BASIC programming language used by the ASCII Unit. Since many of the BASIC commands are nonstandard and peculiar to an ASCII Unit-PC system, it is recommended that even readers already proficient in BASIC pay careful attention to this section.

Section 5 explains the assembly language programming environment and how it relates to the ASCII Unit's BASIC program. It also explains in detail how to write, edit, and run an assembly language program.

Section 6 presents programming examples that are meant to bring together all of the concepts presented in this manual. Most of the programs deal with data transfer and illustrate how the ASCII Unit and the PC work together in various applications. Also in this section are several examples used to illustrate the execution sequence of the hardware during execution of the ASCII Unit and PC programs.

Detailed technical information not immediately necessary for the understanding of a particular section has been put into one of the seven appendices and should be used for reference when needed. For a list of the appendices, see the table of contents.

Note In this manual, ladder diagram instructions are given by mnemonics with the function codes in parentheses following them. The first function code is for C-series PCs and the second function code is for CV-series PCs. For example, in MOV(21/030) (the MOVE instruction), the function code for C-series PCs is 21; for CV-series PCs, 030.

⚠WARNING Failure to read and understand the information provided in this manual may result in personal injury or death, damage to the product, or product failure. Please read each section in its entirety and be sure you understand the information provided in the section and related sections before attempting any of the procedures or operations given.

TABLE OF CONTENTS

PRECAUTIONS

This section provides general precautions for using the Programmable Controller (PC) and related devices.

The information contained in this section is important for the safe and reliable application of the PC. You must read this section and understand the information contained before attempting to set up or operate a PC system.

1 Intended Audience

This manual is intended for the following personnel, who must also have knowledge of electrical systems (an electrical engineer or the equivalent).

- Personnel in charge of installing FA systems.
- Personnel in charge of designing FA systems.
- Personnel in charge of managing FA systems and facilities.

2 General Precautions

The user must operate the product according to the performance specifications described in the operation manuals.

Before using the product under conditions which are not described in the manual or applying the product to nuclear control systems, railroad systems, aviation systems, vehicles, combustion systems, medical equipment, amusement machines, safety equipment, and other systems, machines, and equipment that may have a serious influence on lives and property if used improperly, consult your OMRON representative.

Make sure that the ratings and performance characteristics of the product are sufficient for the systems, machines, and equipment, and be sure to provide the systems, machines, and equipment with double safety mechanisms.

This manual provides information for programming and operating OMRON PCs. Be sure to read this manual before attempting to use the software and keep this manual close at hand for reference during operation.

⚠ WARNING It is extreme important that a PC and all PC Units be used for the specified purpose and under the specified conditions, especially in applications that can directly or indirectly affect human life. You must consult with your OMRON representative before applying a PC System to the abovementioned applications.

3 Safety Precautions

⚠ WARNING Do not attempt to take any Unit apart while the power is being supplied. Doing so may result in electric shock.

⚠ WARNING Do not touch any of the terminals or terminal blocks while the power is being supplied. Doing so may result in electric shock.

⚠ WARNING Do not attempt to disassemble, repair, or modify any Units. Any attempt to do so may result in malfunction, fire, or electric shock.

4 Operating Environment Precautions

⚠ Caution Do not operate the control system in the following locations:

- Locations subject to direct sunlight.
- Locations subject to temperatures or humidity outside the range specified in the specifications.
- Locations subject to condensation as the result of severe changes in temperature.

- Locations subject to corrosive or flammable gases.
- Locations subject to dust (especially iron dust) or salts.
- Locations subject to exposure to water, oil, or chemicals.
- Locations subject to shock or vibration.

⚠ Caution Take appropriate and sufficient countermeasures when installing systems in the following locations:

- Locations subject to static electricity or other forms of noise.
- Locations subject to strong electromagnetic fields.
- Locations subject to possible exposure to radioactivity.
- Locations close to power supplies.

⚠ Caution The operating environment of the PC system can have a large effect on the longevity and reliability of the system. Improper operating environments can lead to malfunction, failure, and other unforeseeable problems with the PC system. Be sure that the operating environment is within the specified conditions at installation and remains within the specified conditions during the life of the system.

5 Application Precautions

Observe the following precautions when using the PC system.

⚠ WARNING Always heed these precautions. Failure to abide by the following precautions could lead to serious or possibly fatal injury.

- Always ground the system to 100 Ω or less when installing the Units. Not connecting to a ground of 100 Ω or less may result in electric shock.
- Always turn OFF the power supply to the PC before attempting any of the following. Not turning OFF the power supply may result in malfunction or electric shock.
  - Mounting or dismounting I/O Units, CPU Units, Memory Units Power Supply Units, or any other Units.
  - Assembling the Units.
  - Setting DIP switches or rotary switches.
  - Connecting cables or wiring the system.
  - Connecting or disconnecting the connectors.

⚠ Caution Failure to abide by the following precautions could lead to faulty operation of the PC or the system, or could damage the PC or PC Units. Always heed these precautions.

- Fail-safe measures must be taken by the customer to ensure safety in the event of incorrect, missing, or abnormal signals caused by broken signal lines, momentary power interruptions, or other causes.
- Always use the power supply voltages specified in this manual. An incorrect voltage may result in malfunction or burning.
- Take appropriate measures to ensure that the specified power with the rated voltage and frequency is supplied. Be particularly careful in places where the power supply is unstable. An incorrect power supply may result in malfunction.
- Install external breakers and take other safety measures against short-circuiting in external wiring. Insufficient safety measures against short-circuiting may result in burning.

- Do not apply voltages to the Input Units in excess of the rated input voltage. Excess voltages may result in burning.
- Do not apply voltages or connect loads to the Output Units in excess of the maximum switching capacity. Excess voltage or loads may result in burning.
- Disconnect the functional ground terminal when performing withstand voltage tests. Not disconnecting the functional ground terminal may result in burning.
- Be sure that all the mounting screws, terminal screws, and cable connector screws are tightened to the torque specified in this manual. Incorrect tightening torque may result in malfunction.
- Leave the label attached to the Unit when wiring. Removing the label may result in malfunction if foreign matter enters the Unit.
- Remove the label after the completion of wiring to ensure proper heat dissipation. Leaving the label attached may result in malfunction.
- Double-check all wiring and switch settings before turning ON the power supply. Incorrect wiring may result in burning.
- Wire correctly. Incorrect wiring may result in burning.
- Mount Units only after checking terminal blocks and connectors completely.
- Be sure that the terminal blocks, Memory Units, expansion cables, and other items with locking devices are properly locked into place. Improper locking may result in malfunction.
- Check the user program for proper execution before actually running it on the Unit. Not checking the program may result in an unexpected operation.
- Confirm that no adverse effect will occur in the system before attempting any of the following. Not doing so may result in an unexpected operation.
    - Changing the operating mode of the PC.
    - Force-setting/force-resetting any bit in memory.
    - Changing the present value of any word or any set value in memory.
- Resume operation only after transferring to the new CPU Unit the contents of the DM Area, HR Area, and other data required for resuming operation. Not doing so may result in an unexpected operation.
- Do not pull on the cables or bend the cables beyond their natural limit. Doing either of these may break the cables.
- Do not place objects on top of the cables or other wiring lines. Doing so may break the cables.
- Use crimp terminals for wiring. Do not connect bare stranded wires directly to terminals. Connection of bare stranded wires may result in burning.
- When replacing parts, be sure to confirm that the rating of a new part is correct. Not doing so may result in malfunction or burning.
- Before touching a Unit, be sure to first touch a grounded metallic object in order to discharge any static built-up. Not doing so may result in malfunction or damage.

SECTION 1
Hardware

This section describes the external hardware of the ASCII Unit. The front and back panels of the ASCII Unit contain switches, buttons, connectors, and indicators which enable the user to setup, control, and monitor ASCII Unit operations. The ASCII Unit's internal configuration as well as a typical system configuration are also illustrated.

1-1 Front Panel

On the front panel of the ASCII Unit, there are six indicator lights, the reset switch, the START/STOP switch, two RS-232C connectors, and a battery compartment. In addition, behind the LED Display Panel, is an 8-pin DIP switch used for setting various control parameters.

Ports

The front panel of the ASCII Unit contains two RS-232C ports. These ports are used for connecting peripheral I/O devices to the ASCII Unit. Both ports can be used for communication devices such as printers, terminals, and modems. Only port 1 can be used for uploading or downloading a BASIC program. The standard configuration is a personal computer connected to port 1 and a printer or other I/O device connected to port 2.

Switches

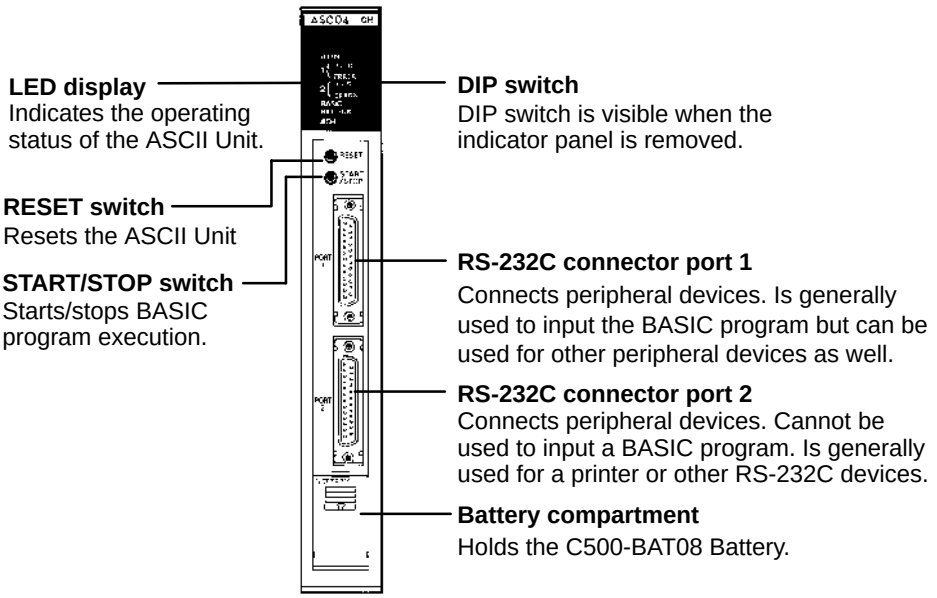
The **START/STOP** switch is a toggle switch and is used for initiating and halting execution of the ASCII Unit program.

The **RESET** switch is used for resetting the ASCII Unit.

Battery Compartment

The battery compartment holds the C500-BAT08 Battery.

Front Panel



Indicator LEDs

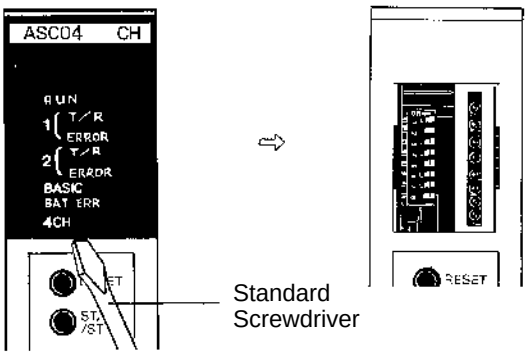
The following table describes the ASCII Unit's indicators.

Name	Indication	Function
RUN	Lit (green)	Lit when the ASCII Unit is operating normally. Unlit if an error occurs.
T/R for ports 1 and 2	Blinking (green)	Blinks during data transmission (port 1 and port 2).
ERROR 1 (port 1) ERROR 2 (port 2)	Lit (red)	Lit if a reception buffer overflows or an error such as parity error occurs (see note), or while the ASCII Unit is waiting for specific transmission conditions to be satisfied.
BASIC	Lit (green)	Lit while the BASIC program is running.
	Blinking (green)	Blinks when the BASIC program stops, or when the ASCII Unit is waiting for input while the BASIC program is running.
	Unlit (green)	Unlit when in monitor mode.
BAT ERR	Blinking (red)	Blinks when the battery voltage has fallen below the rated level or if the battery is not inserted correctly.
4CH	Lit (green)	Lit when the ASCII Unit is set for 4-word mode. Unlit when the ASCII Unit is set for 2-word mode.

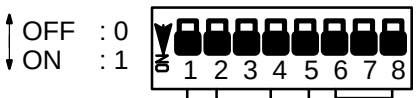
Note When a reception buffer overflows or transmission error occurs, the red indicator is lit and will not be turned off even if the transmission error or reception buffer overflow is corrected, because the error log must be kept. To turn off the indicator, execute the CLOSE instruction or stop the program.

Front Panel DIP Switch

In order to access the front panel DIP switch, the indicator cover must be removed with a standard screwdriver as shown in the illustration below. To set the DIP switch, the power to the ASCII Unit must be OFF. The DIP switch must be set before the ASCII Unit is mounted to the PC. Make sure the power to the PC is off when mounting the ASCII Unit.



DIP Switch Settings



Start mode

Pin No.	1	Function
Setting	0	Manual start mode In this mode, the BASIC program is not started upon power application. To start the program, either press the START/STOP switch or issue a start command from the personal computer connected to port 1.
	1	Automatic start mode In this mode, the BASIC program is started automatically on power application.

Automatic program transfer from EEPROM to RAM

Pin No.	2	Function
Setting	0	Set this pin to “0” if only the RAM is to be used.
	1	Set this pin to “1” to automatically transfer the program from the EEPROM to RAM on power application or reset.

Program No.

These pins select which program will be executed on power application or reset. The program number can be changed later with the PGEN command.

Pin No.	3	4	Function
Setting	0	0	No. 1
	1	0	
	0	1	No. 2
	1	1	No. 3

Screen size

Pin No.	6	7	8	Screen Size
Setting	0	0	0	40 columns x 7 lines
	1	0	0	40 columns x 8 lines
	0	1	0	40 columns x 15 lines
	1	1	0	40 columns x 16 lines
	0	0	1	80 columns x 15 lines
	1	0	1	80 columns x 16 lines
	0	1	1	80 columns x 24 lines
	1	1	1	80 columns x 25 lines

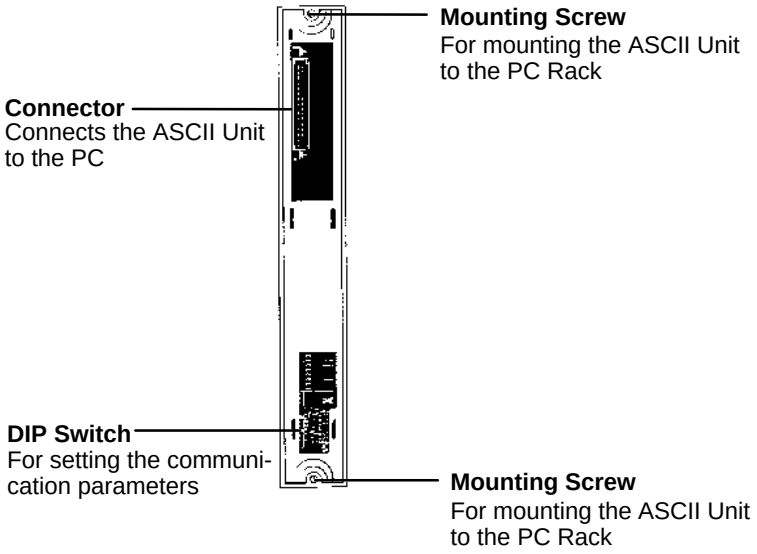
Specifies 2 or 4 word setting for the Data Section.

Pin No.	5	Function
Setting	0	Two word setting. Choose this setting to use WRIT(87/191)/READ(88/190)
	1	Four word setting. This setting is used when the ASCII Unit is mounted to a Slave Rack or when the PC does not support WRIT(87/191)/READ(88/190).

1-2 Back Panel

The back panel of the ASCII Unit houses the PC connector and an 8-pin DIP switch used for setting the communication parameters.

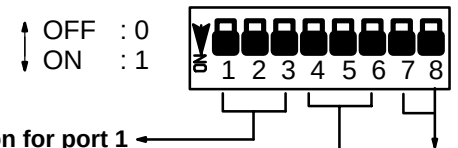
Back Panel



Back Panel DIP Switch

- Pins 1, 2, and 3 are used for setting the baud rate of port 1.
- Pins 4, 5, and 6 are used for setting the baud rate of port 2.
- Pins 7 and 8 are not used but must be set to OFF. If they are left ON, the Hardware Test program will be executed and all RAM data will be lost.

The DIP switch settings are described in more detail in the following diagram.



Baud rate selection for port 1

Pin No.	1	2	3	Baud Rate
Setting	0	0	0	300 bps
	1	0	0	600 bps
	0	1	0	1,200 bps
	1	1	0	2,400 bps
	0	0	1	4,800 bps
	1	0	1	9,600 bps
	0	1	1	
	1	1	1	

Not used (Always set these pins to OFF.)

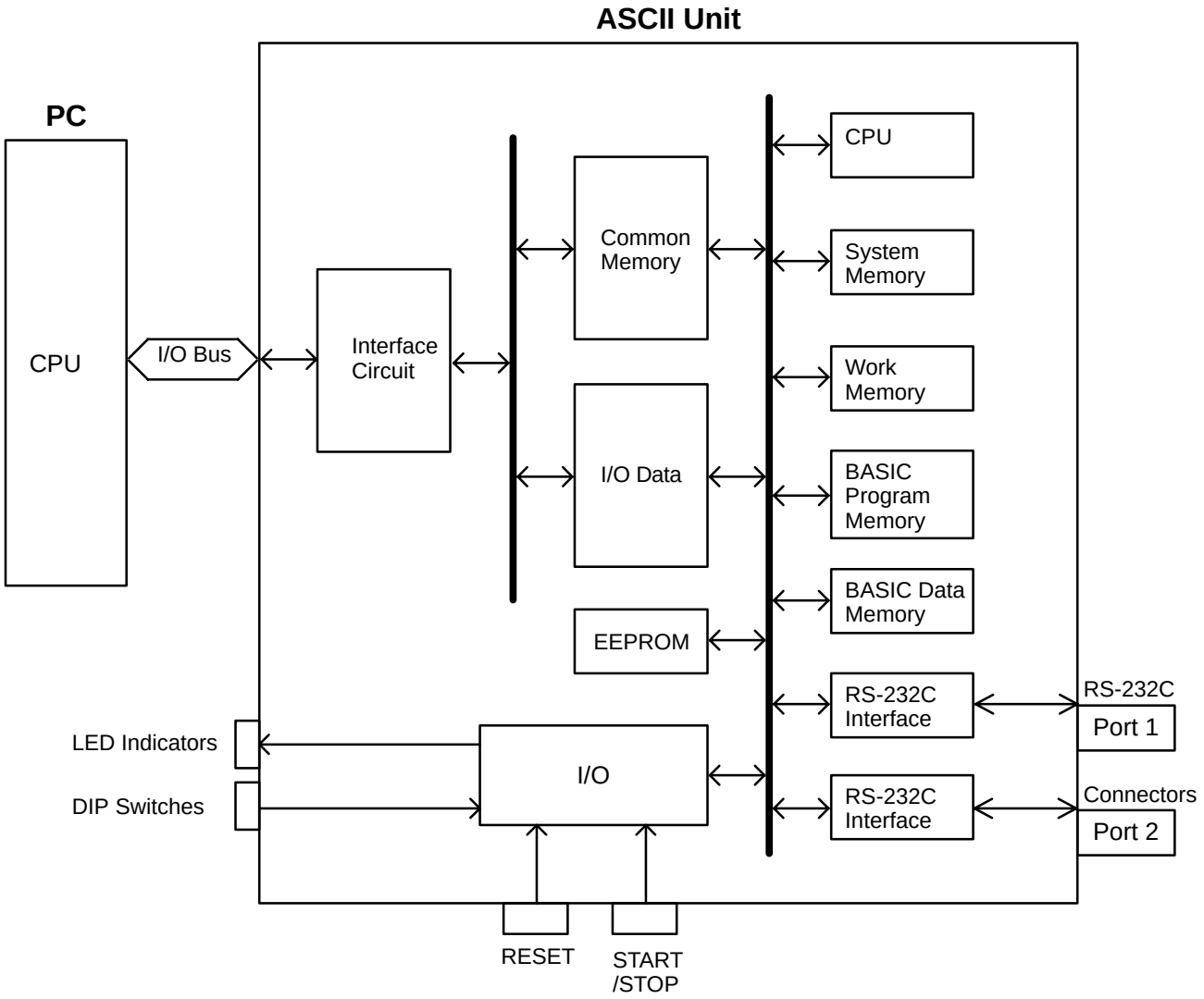
Baud rate selection for port 2

Pin No.	4	5	6	Baud Rate
Setting	0	0	0	300 bps
	1	0	0	600 bps
	0	1	0	1,200 bps
	1	1	0	2,400 bps
	0	0	1	4,800 bps
	1	0	1	9,600 bps
	0	1	1	19,200 bps
	1	1	1	

1-3 ASCII Unit Internal Configuration

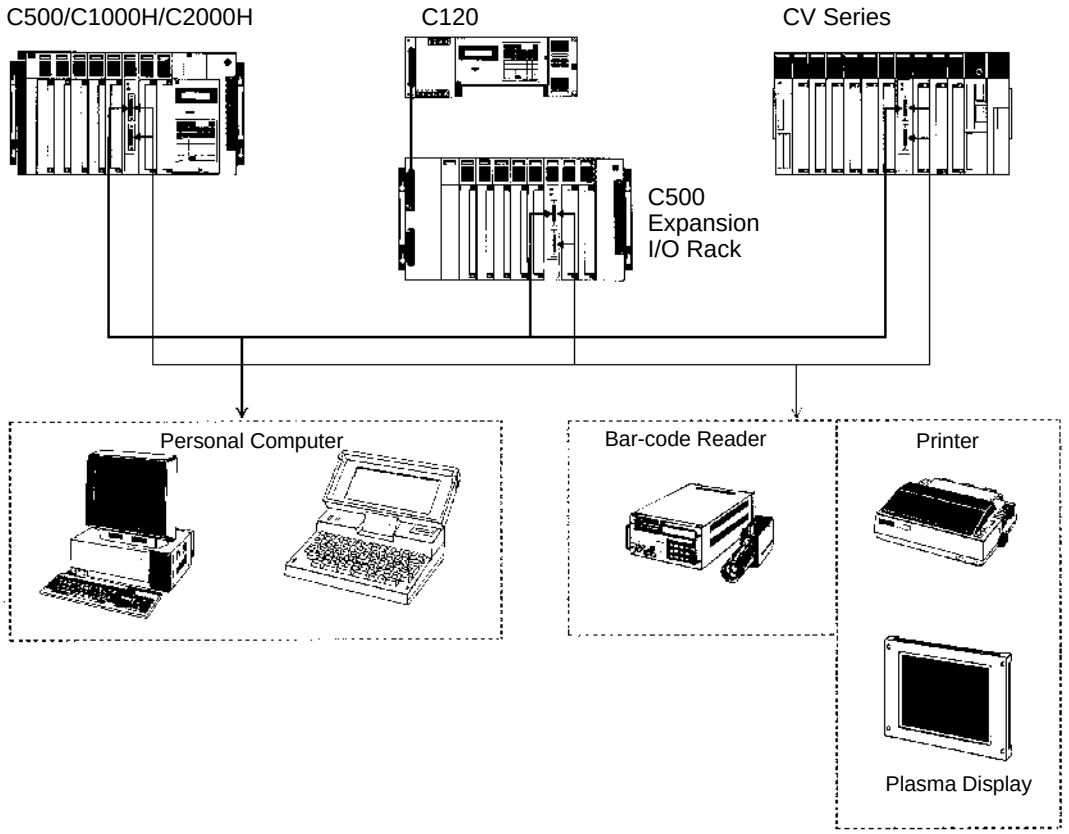
The Common Memory can be accessed using the ASCII Unit's PC READ or PC WRITE statements. It can also be accessed using the PC's WRIT(87/191) and READ(88/190) instructions. I/O data can be accessed using the ASCII Unit's PC GET, PC PUT, and ON PC statements. It can also be accessed using the MOV(21/030) instruction.

The following figure illustrates these instructions and their relationship to the Common Memory and the I/O data.



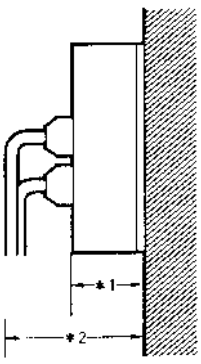
1-4 System Configuration

The ASCII Unit can be mounted to any slot on the CPU Backplane. Before mounting the ASCII Unit, the DIP switches must be set. Make sure that the power supply to the PC is turned OFF during installation of the ASCII Unit. A personal computer used for entering the BASIC program should be connected to Port 1 and other peripheral I/O devices such as a printer or a display terminal can be connected to Port 2 (refer to the following diagram). For more detailed information on peripheral interface connections and timing, refer to *Appendix B*.



1-5 Mounting

The ASCII Unit can be mounted to any I/O slot. The control panel must allow enough space for the connectors, as shown in the figure below.



*1. Height of the ASCII Unit including the base (100 mm)

*2. Height of the ASCII Unit with an RS-232C connector attached (approximately 160 to 180 mm)

SECTION 2
Data Allocations

This section explains the words of the PC used to communicate with the ASCII Unit.

2-1 Bits and Words

The PC's memory is divided up into many sections, each of which has its own unique name and purpose. The ASCII Unit can access any of these memory areas using the BASIC READ and WRITE statements (this is explained in more detail in *Section 4 BASIC Programming*). However, there are words in the PC's IR data area that are uniquely assigned to each ASCII Unit.

The PC's memory is organized into units called *words*. Information is usually stored in word or multiple word units. Each word has a unique address in the computer memory and can be accessed by specifying its address.

Each word contains 16 bits. A bit is the smallest piece of information that can be stored or accessed by a computer. A bit is always in one of two states: zero or one (OFF or ON). Certain bits can be accessed individually and are used as flags. A *flag* is turned ON and OFF by the hardware to indicate some state of the computer or to enable or disable certain operations. Bits can also be set or cleared by the programmer to communicate certain parameters or conditions to the CPU.

For example, when the ASCII Unit program requests data to be sent from the PC using the BASIC GET statement, the PC's Write Flag is turned OFF, indicating that the ASCII Unit must wait while the PC prepares the data. When the PC has collected the data, it turns ON the Write Flag, signaling the ASCII Unit that it may proceed to read the data.

2-2 Data Configuration

Each ASCII Unit is assigned a section of memory in the PC. The data has two configurations, *two-word* and *four-word*. The data configuration is selected by setting pin 5 of the front panel DIP switch before power is applied to the ASCII Unit.

The basic difference between the two-word and four-word configurations is that in two-word mode the WRIT(87/191)/READ(88/190) instructions are supported for data transfer while in four-word mode they are not supported. The structure and application of the words in each of the two modes is explained next.

2-2-1 Two-word Configuration

WRIT(87/191) and READ(88/190) are supported in the two-word configuration. WRIT(87/191) is the PC's I/O WRITE instruction and READ(88/190) is the PC's I/O READ instruction.

When the PC uses these instructions for data transfer, up to 255 words of data can be transferred at one time. In order to transfer multiple data words at the same time, however, the ASCII Unit must be programmed to use the PC READ or PC WRITE statements. In addition the A or S formats must be used. Refer to *Appendix D* for more information on formats.

The following PCs support WRIT(87/191)/READ(88/190):
C500: 3G2C3-CPU11-EV1
C120: 3G2C4-SC024-EV1
All C1000H, C2000H, CV-series PCs.

When WRIT(87/191)/READ(88/190) are not supported or not used, data is transferred using the PC's MOV(21/030) instruction. When the MOV(21/030) is used, only one word of data is transferred at a time.

To output (word n) data using the MOV(21/030), set bits 00, 01, 02 and 03 to zero.

Data Bit Definitions

The following table identifies the individual bits in the two words allocated to the ASCII Unit. In the following Bit Definition table, entries in the *Bit* column enclosed

in parentheses are reserved for use by WRIT(87/191)/READ(88/190) and are not available for general programming application.

Word	Bit	Function	Description
n	(00)	PC busy	Reserved for WRIT(87/191)/READ(88/190)
	(01)	PC WRITE complete	
	(02)	PC READ complete	
	03	Restart	The ASCII Unit is activated when this bit goes OFF
	04 to 07	---	Not Used
	08 to 15	Output data bits 0 to 7	Data output from the PC to the ASCII Unit. Read by the PC GET statement.
n+1	(00)	ASCII busy	Reserved for WRIT(87/191)/READ(88/190)
	(01)	PC READ complete	
	(02)	PC WRITE complete	
	03	ASCII error	Turns ON when an error occurs in the ASCII Unit, when the RESET activates, or when the ASCII Unit restarts.
	04	Port 1 error	Turns ON when a buffer overflows or transmission error occurs in Port 1. Turns OFF when the CLOSE statement is executed or the program is stopped.
	05	Port 2 error	Turns ON when a reception buffer overflows or transmission error occurs in Port 2. Turns OFF when the CLOSE statement is executed or the program is stopped.
	06	Battery error	Turns ON when the battery is low or removed
	07	BASIC RUN	Turns ON when a BASIC program is running
	08 to 15	Input data bits 0 to 7	Data output from the ASCII Unit to the PC. Written by the PC PUT statement.

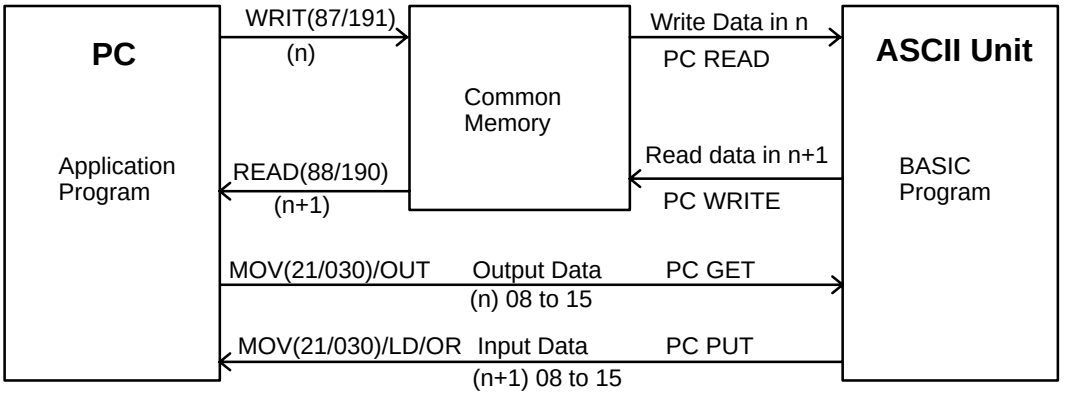
Note When the reset switch is turned ON, the data in word n+1 will be $FFF9. Restarting can be checked using bit 03 of word n+1.
When the ASCII Unit is restarted, the data of word n+1 will be 0000.

Program Execution

The following diagram illustrates how the words and bits allocated to the ASCII Unit relate to program execution.

WRIT(87/191) is executed when the data communication condition for WRIT(87/191) is satisfied and the ASCII busy flag is cleared. If these conditions are not met, the WRIT(87/191) is treated as a NOP.

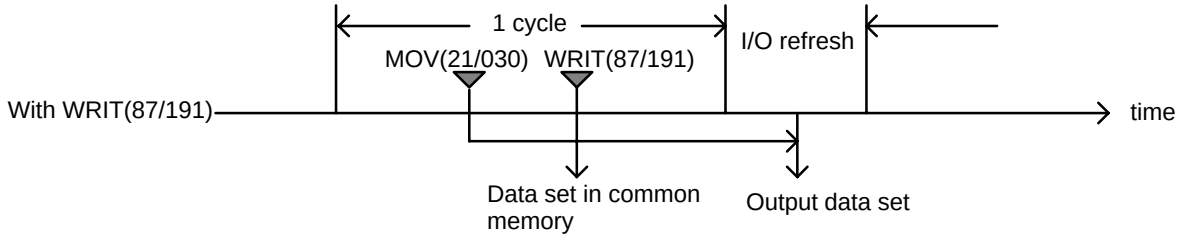
READ(88/190) is executed when the data communication condition is satisfied and the ASCII busy flag and ASCII write complete flag are OFF. If these conditions are not met, the READ(88/190) is treated as a NOP.



Timing

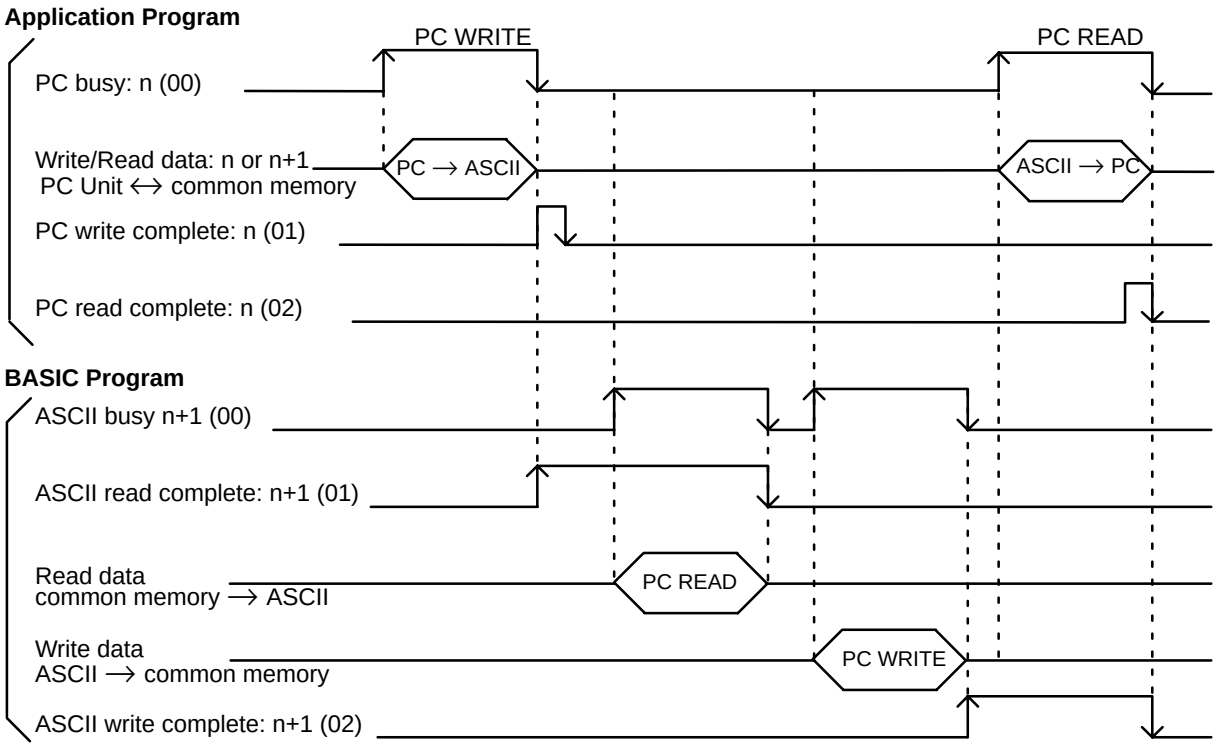
The WRIT(87/191) and READ(88/190) instructions are executed and the common memory is refreshed every time the PC completes one cycle of the program. I/O data, however, does not use the common memory (see above diagram) and is refreshed when the PC refreshes all the I/O data. Consequently

there is a time difference between when common memory data is set and when I/O data is set. This time difference must be taken into consideration when preparing both the ASCII Unit and PC programs.

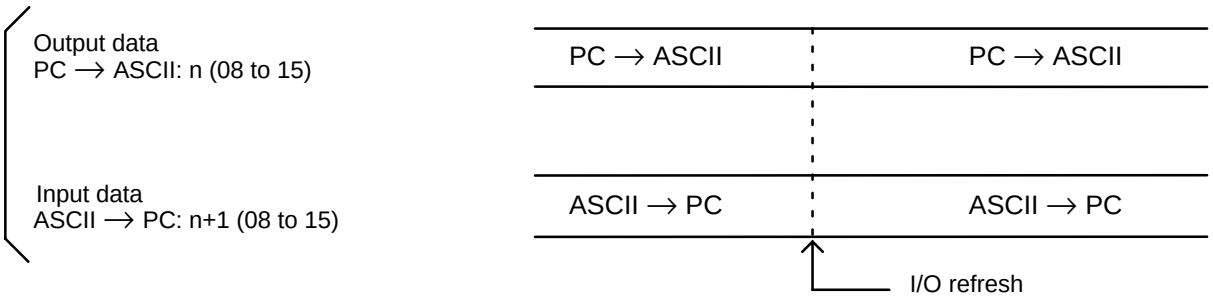


The following diagram illustrates the various timing relationships between the PC and ASCII Unit during data transfer.

Relationship between READ and WRITE Timing



Relationship between Output and Input Timing



2-2-2 Four-Word Configuration

In four-word mode, WRIT(87/191) and READ(88/190) instructions cannot be used. The ASCII Unit can be set to four-word mode by setting pin 5 of the front panel DIP switch to ON.

Bit Allocation

The following two tables identify the individual bits in the four words allocated to the ASCII Unit. Notice that words n and n+1 are used for output and words n+2

and n+3 are used for input. In this case, input and output are from the point of view of the PC.

Bit	Word n (OUT)	Word n+1 (OUT)	Word n+2 (IN)	Word n+3 (IN)
00	Write Data 00	PC busy	Read Data 00	ASCII busy
01	Write Data 01	PC write complete	Read Data 01	ASCII read complete
02	Write Data 02	PC read complete	Read Data 02	ASCII write complete
03	Write Data 03	Restart	Read Data 03	ASCII error
04	Write Data 04	Interrupt No. 00	Read Data 04	Port 1 error
05	Write Data 05	Interrupt No. 01	Read Data 05	Port 2 error
06	Write Data 06	Interrupt No. 02	Read Data 06	Battery error
07	Write Data 07	Interrupt No. 03	Read Data 07	BASIC RUN
08	Write Data 08	Output Data 00	Read Data 08	Input Data 00
09	Write Data 09	Output Data 01	Read Data 09	Input Data 01
10	Write Data 10	Output Data 02	Read Data 10	Input Data 02
11	Write Data 11	Output Data 03	Read Data 11	Input Data 03
12	Write Data 12	Output Data 04	Read Data 12	Input Data 04
13	Write Data 13	Output Data 05	Read Data 13	Input Data 05
14	Write Data 14	Output Data 06	Read Data 14	Input Data 06
15	Write Data 15	Output Data 07	Read Data 15	Input Data 07

Bit Definitions

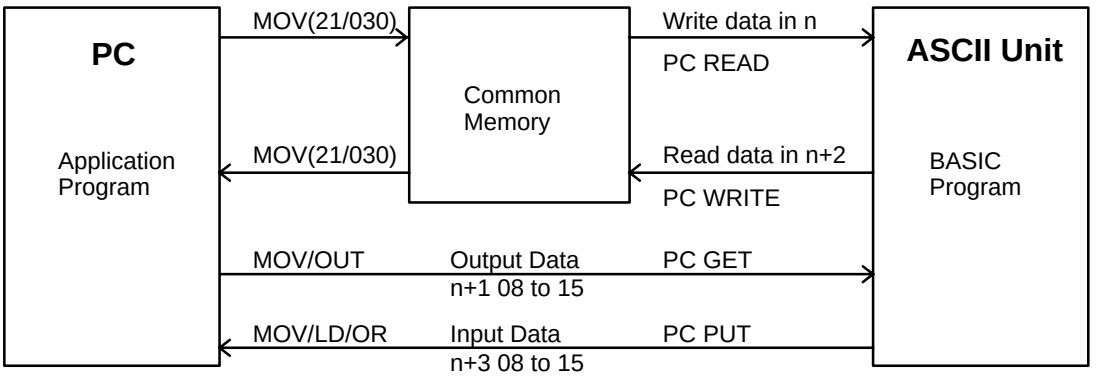
Word	Bit	Function	Description
n	00 to 15	Write data bits 00 to 15	Data that will be written to the common memory from the PC by the MOV(21/030) and read with the PC READ statement.
n+1	00	PC busy	Set by the PC program when the PC accesses common memory, and cleared when memory access is terminated. The ASCII Unit cannot access the common memory while this bit is set.
	01	PC write complete	Momentarily set by the PC program when the PC has completed writing data to the common memory. When this bit goes ON, the ASCII Unit read complete flag n+3 (01) goes ON as well.
	02	PC read complete	Momentarily set by the PC program when the PC has completed reading data from the common memory. When this bit goes ON, the ASCII Unit write complete flag n+3 (02) goes OFF as well.
	03	Restart	The ASCII Unit is activated at the trailing edge of this flag (when the flag goes OFF). A differentiated signal must be used for the Restart signal.
	04 to 07	Interrupt number bits 00 to 03	Serves as an interrupt number when the ON PC statement is used.When bits 00 to 03 are converted into hexadecimal 00 to 15, 00 is ignored and 01 to 15 are used as valid interrupt numbers.
	08 to 15	Output data bits 00 to 07	Data output from the PC to the ASCII Unit, written by the MOV and read with the PC GET statement.
n+2	00 to 15	Read data bits 00 to 15	Data that will be written to the common memory from the ASCII Unit with the PC WRITE statement and read with the MOV.
n+3	00	ASCII busy	Set when the ASCII Unit accesses the common memory and cleared when memory access is terminated. The PC cannot access common memory while this bit is set.
	01	ASCII read complete	Momentarily set when the PC write complete flag goes ON enabling the ASCII Unit to read from common memory. This flag is cleared when the ASCII Unit terminates the read operation.
	02	ASCII write complete	Set at the time the ASCII Unit terminates a write operation to the common memory and cleared when the PC read complete flag goes ON.
	03	ASCII error	Set when an ASCII Unit error occurs, when RESET is activated, or when the ASCII Unit restarts.
	04	Port 1 error	Set when a reception buffer overflows or transmission error occurs at Port 1. Turns OFF when the CLOSE statement is executed or the program is stopped.
	05	Port 2 error	Set when a reception buffer overflows or transmission error occurs at Port 2. Turns OFF when the CLOSE statement is executed or the program is stopped.
	06	Battery error	Set when the battery is low or removed.
	07	BASIC RUN	Set when the BASIC program is running.
	08 to 15	Input data bits 00 to 07	Data written with the PC PUT statement and read with the MOV.

Note

1. Apart from the data used to read bit 00 to 15 of word n+2, the input data of bit 08 to 15 of word n+2 can be used for program control of the PC by transmitting the 8-bit data to the PC.
2. When the reset switch is turned ON, the data in word n+1 will be $FFF9. Restarting can be checked using bit 03 of word n+1.
   When the ASCII Unit is restarted, the data of word n+1 will be 0000.

Program Execution

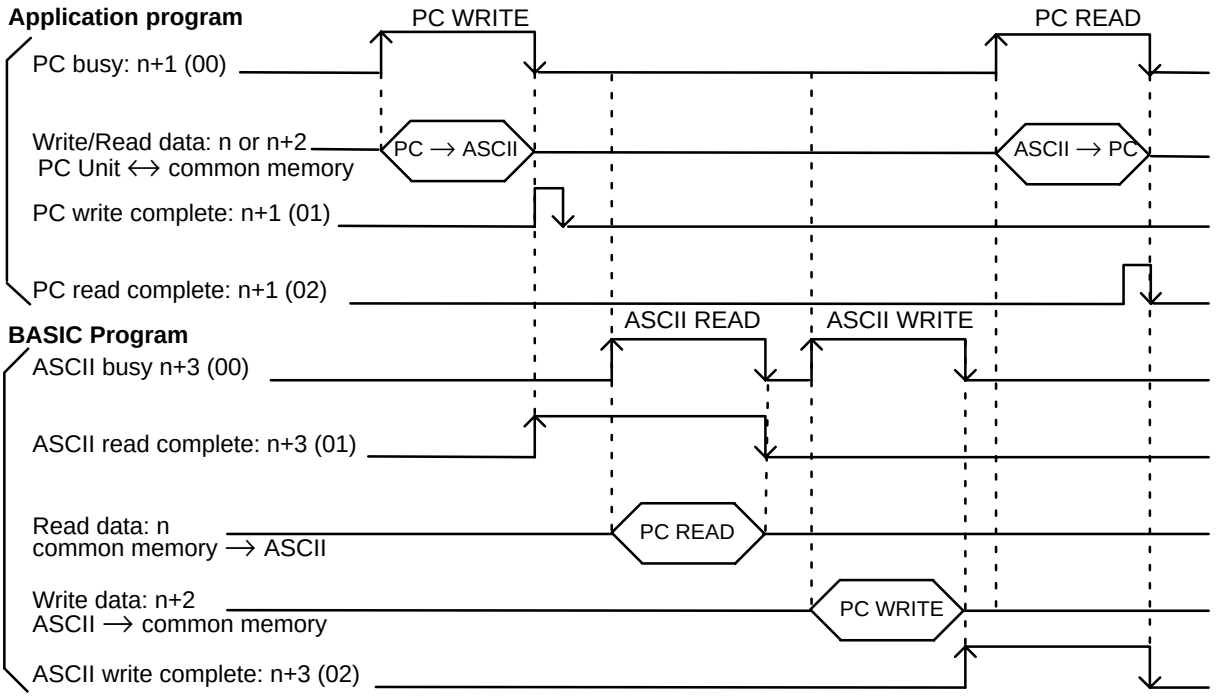
The following diagram illustrates how the words and bits allocated to the ASCII Unit relate to program execution.



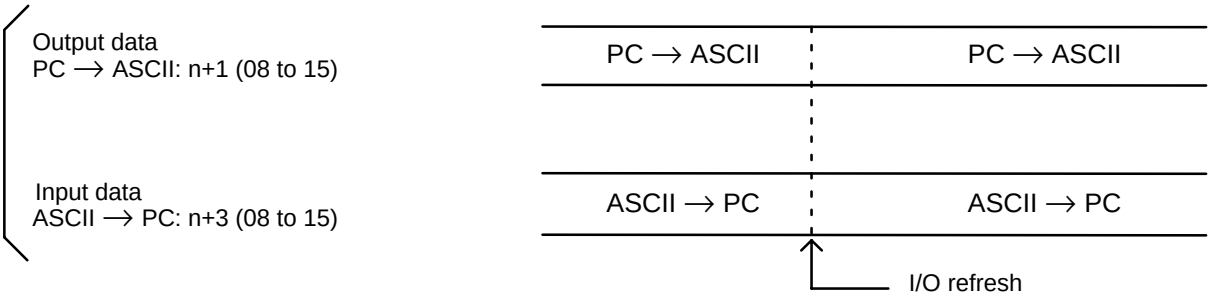
Timing

The following diagram illustrates the various timing relationships between the PC and ASCII Unit during data transfer.

Relationship between READ and WRITE Timing



Relationship between Output and Input Timing



SECTION 3
Programming and Communications

The first part of this section explains how the ASCII Unit and the PC exchange information.

The second part of this section explains how to transfer programs from one device to another. The ASCII Unit's BASIC program is written on a personal computer. To run the program, it must be transferred to the RAM of the ASCII Unit. The ASCII Unit program can be permanently stored in the ASCII Unit's EEPROM and also loaded from the EEPROM. The program can also be transferred back to the personal computer or other storage device.

The last part of this section explains how to run a BASIC program once it has been transferred to the ASCII Unit.

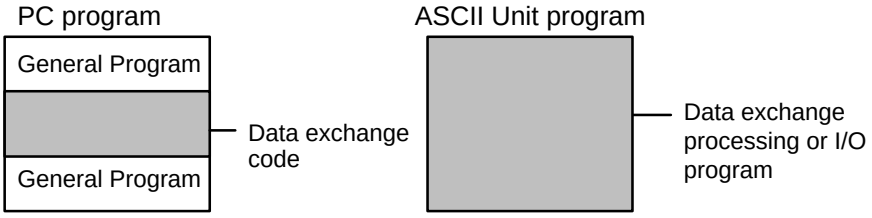
3-1 Programs

To use the ASCII Unit in conjunction with the PC, an ASCII Unit program written in BASIC is needed. A data exchange routine must also be incorporated into the PC program. The PC data exchange routine must set the number of words to be transferred, the base address, and the specific memory area. This can be done using the PC's MOV(21/030) instruction.

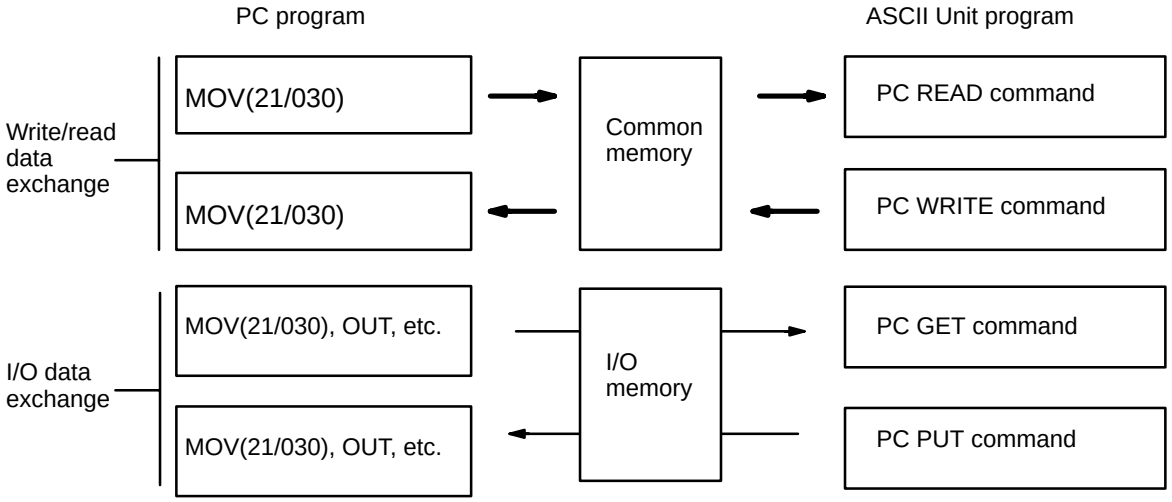
There are two ways the ASCII Unit can communicate with the PC. In the first method, the PC controls the timing of the data transfer between the two devices. The ASCII Unit "requests" access to the PC data memory area using the PC READ, PC WRITE, PC GET, or PC PUT statements, and then waits for the PC to respond by setting either the read or write flag. The PC data exchange routine performs the designated operations. When the PC is ready, the appropriate flag is set and the ASCII Unit proceeds with the data transfer.

In the second method, the WRIT(87/191) and READ(88/190) instructions are used in conjunction with the PC READ, PC WRITE, PC GET, and PC PUT statements to transfer data.

This diagram illustrates the PC and ASCII Unit programs.



This diagram illustrates the relationship between the PC data exchange code and the ASCII Unit program.



3-2 Program Transfer

Preparation

For the personal computer to communicate with the ASCII Unit, set the computer communication software as follows:

Baud rate: same as ASCII Unit
Data length: 8 bits
Parity: none
No. stop bits: 2

Also: Full duplex, no echo, no XON/XOFF buffer busy control, no auto line feed.

Set the ASCII Unit DIP switches to the desired configuration (refer to *Section 1 Hardware*).

Transfer

The ASCII Unit's BASIC or assembly language program must be written on a personal computer which is connected to port 1 of the ASCII Unit through an RS-232C interface. A program can be transferred to the ASCII Unit from the personal computer or any other storage device connected to one of the communication ports with the BASIC LOAD command or the S and L commands. Programs can also be transferred from the ASCII Unit's EEPROM to the ASCII Unit's RAM using the LOAD command.

Programs can be transferred from the ASCII Unit's RAM to the EEPROM or to a personal computer or other storage device connected to one of the communication ports using the BASIC SAVE command.

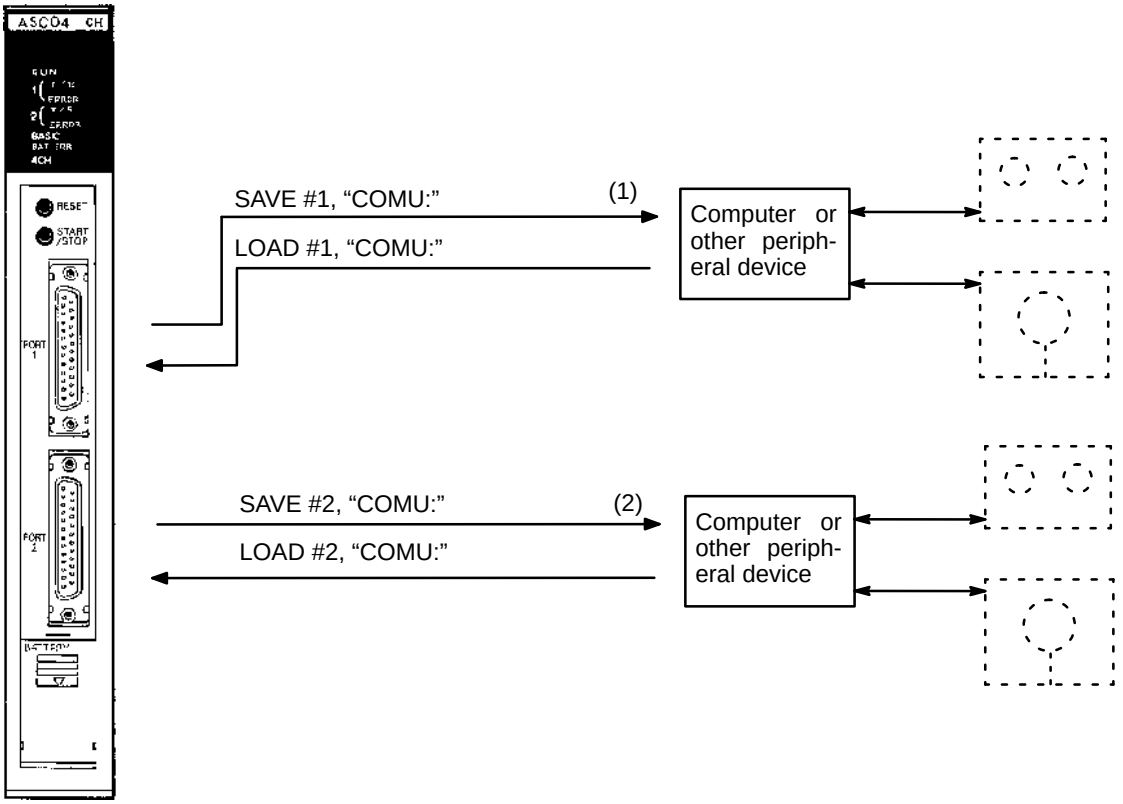
The ASCII Unit can be booted on power up by a program stored in the EEPROM. To do this set pin 2 of the front panel DIP switch on the ASCII Unit to ON.

Note

1. During data transfer, an overflow may occur if the buffering capacity of the baud rate settings of the computer and the ASCII Unit are not matched. If an overflow error does occur, set either a slower baud rate or specify XON with the OPEN command.
2. Programs named with PNAME cannot be transferred. Delete the name by executing PNAME " " if necessary before attempting to transfer a program.

The FIT or LSS can be used to back up BASIC programs onto floppy disks, consult the FIT or LSS Operation Manual.

The following figure illustrates the direction of data transfer when using the SAVE and LOAD commands.



Note

1. The EEPROM'sn lifetime is limited to 5,000 write operations.
2. Refer to the explanation of the OPEN statement for details on COMU.

3-3 Running the BASIC Program

The ASCII Unit can store and access three separate BASIC programs. Each program has an associated program number. The user can specify which program is to be used by setting pins 3 and 4 of the front panel DIP switch. This must be done before the Unit is activated.

There are three ways to execute the specified BASIC program:

- Enter the RUN command from the keyboard of the personal computer. (Keying in CTRL+X will abort the program.)
- Press the START/STOP switch. Press it again to stop the program.
- If pin 1 of the front panel DIP switch is set to the ON position, the specified program will be executed automatically when the Unit is turned ON or reset.

3-4 Assembly Routines

Use the monitor mode of the ASCII Unit for writing assembly language routines to execute operations that cannot be processed with BASIC programs. The ASCII Unit incorporates the Hitachi HD6303 CPU.

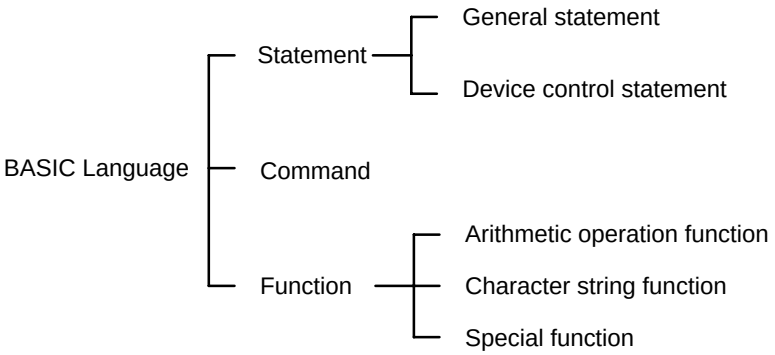
Assembly language routines can be written for the ASCII Unit and called from the BASIC program with the USR statement. An assembly program can be saved to the personal computer with the S command and loaded from the personal computer with the L command. Assembly programs are stored in the S format.

SECTION 4
BASIC Programming

This section contains an explanation of the terminology, components, structure, and use of the BASIC programming language on the ASCII Unit. Even those familiar with BASIC should study this section carefully as many of the ASCII Unit BASIC commands, statements, and functions are non-standard, especially those that control I/O operations. Readers should pay special attention to the explanations of statements that are prefixed with "PC." Also pay special attention to the OPEN statement.

4-1 Program Configuration

A BASIC program consists of commands, statements, and functions.



Basic Statements designate and control the flow of programs and are generally used in program lines within a program. Statements are usually created as programs and executed by the RUN command. Statements can be directly input and executed from the keyboard.

Basic Commands are usually entered from the command line and control operations external to the program such as printing and listing. Commands must be directly input and executed from the keyboard. Commands cannot be inserted into programs and executed by the RUN command. If commands are inserted into programs and executed, the commands may not work properly.

Examples: print, list, run

Functions are self-contained programs which accept one or more arguments, perform predefined calculations, and return a result(s). There are predefined BASIC functions for arithmetic and string operations as well as user defined functions.

Examples: INT(x), LOG(x), SQR(x)

Lines and Statements

A program written in BASIC is a series of lines, each of which consists of one or more statements. If several statement are written on the same line, they must be separated with colons(:). A line can be no longer than 255 characters. Use single quote marks (') to separate comments.

Example of four statements on a line:

10 FOR L=1 TO 100: J=L*I: PRINT J: NEXT L

Line Numbers

Every BASIC program line begins with a line number. Line numbers indicate the order in which the program lines are stored in memory and are also used as references for branching and editing. Line numbers must be in the range of 0 to 63999. A period may be used in AUTO, DELETE, EDIT, and LIST commands to refer to the current line.

Examples: LIST. EDIT. AUTO DEL 100–

Character Set

The BASIC character set is comprised of alphabetical characters, numeric characters, and special characters.

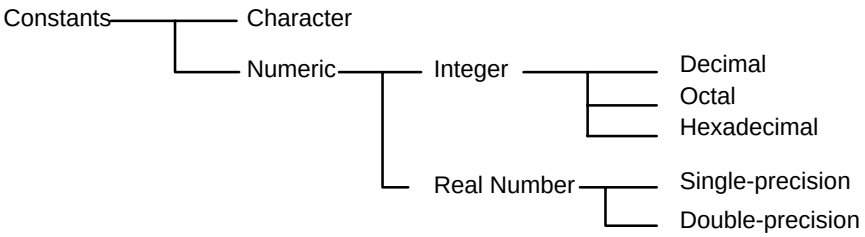
The alphabetic characters in BASIC are the upper case and lower case letters of the alphabet. The numeric characters in BASIC are the digits 0 through 9.

The following special characters are recognized by BASIC:

SP (space) ! " # $ & ' ( ) * + , – . / : ; < = > ? [\ } ^ _

Constants

The following can be used as constants:



Character Constants

A character constant is a character string enclosed by double quotation marks (”). It can be up to 255 characters long. If it has no character, it is called an “empty character string” or a null string.

Example: “CF-BASIC”

Integers Constants

Whole numbers between –32768 and 32767 can be used. An optional percent sign (%) can be added to specifically indicate an integer constant. Integer constants do not have decimal points.

Examples: 1234 –1234 12

Octal Constants

Octal numbers from 0 to 7 beginning with the prefix “&” and within the range of &0 to &177777 can be used.

Examples: &0127 &7777

Hexadecimal Constants

Hexadecimal numbers with the prefix “&H”, from 0 to F (0 to 9,A,B,C,D,E,F) and in the range &H0000 to &HFFFF can be used.

Examples: &H5E &HBF4

Floating Point Constants

Single precision: This type of constant is stored with seven-digit precision and is output as a six-digit constant with the seventh digit rounded off. It is represented by one of the following methods:

1, 2, 3...

1. As a number with seven or less digits: 1234.5
2. As a number in exponential form using E: 1.2E+3
3. As a number with the character “!” at the end: 2.34!

Double precision: This type of constant is stored with 16-digit precision and is output as 16 digits or less. It is represented by one of the following methods:

1, 2, 3...

1. As a number with 8 or more valid digits: 1.23456789
2. As a number in exponential form using D: –1.2D–3
3. As a number with the character “#” at the end: 2.34#

Variables

Variables are names used to represent values that are used in a BASIC program. The value of a variable may be assigned as the result of calculations or explicitly by the programmer with an assignment statement. If no value is assigned to a numeric variable, it is assumed to be zero. If no value is assigned to a character variable, it is assumed to be a null string.

Variable Name

A variable may be up to 255 alphanumeric characters long, but only the first 16 characters are actually valid. No variable can start with “FN” or a valid BASIC command name.

Note A syntax error will occur if a variable begins with a reserved word (i.e., in the case of TOTAL or ABSOL, a syntax error will occur because TO and ABS are reserved words).

Type Declarator

The variable TYPE must be declared. This is done using a type declarator which is placed after the variable name. Even if two variables have the same name, they will be treated differently if they are declared as different types of variables.

Integer: Uses 2 bytes per variable.

! Single-precision real: Uses 4 bytes per variable.

Double-precision real: Uses 8 bytes per variable.

$ Character: Uses a maximum of 255 characters.

There is a second way to declare variable types. The BASIC statements DEFINT, DEFSTR, DEFSNG, and DEFDBL may be used to declare the types for certain variable names.

Variable Array

An array is a group of values of the same TYPE that is stored and referenced as a unit by the same variable name. Each element in an array has a unique position and is referenced by the name of the array subscripted with an integer or integer expression.

There can be many dimensions to an array. The most common types are one, two, and three dimensional arrays. An array has one subscript for each dimension in the array.

For example, T(4) would reference the fourth element in the one-dimensional array T. R(2,3) would reference the value located in the second row and third column of the two-dimensional array R.

The maximum number of dimensions of an array is 255. The maximum number of elements per dimension is 32767. The array size and number of dimensions must be declared with the DIM statement. The subscript value zero is the position of the first element in an array. All elements of an array must be of the same TYPE.

Type Conversion

When necessary, BASIC will convert a numeric constant from one TYPE to another. The following rules and examples apply:

1, 2, 3...

1. If the numeric data on the right side of an assignment statement differs from the type of data on the left side, the right side is converted to match the left. However, character data cannot be converted to numerical data, or vice versa.

   Example: A = 12.3: if A is an integer then, “12” is assigned to A.

2. Double-precision data is converted to single-precision data when assigned to a single-precision variable.

   Example:

   IF “A” is a single-precision variable and the statement:

   LET A = 12.3456789# occurs in a program, then 12.3456789# will be converted to a single-precision number and then assigned to “A.”

3. When an arithmetic operation is performed using both single-precision and double-precision values, the single-precision value is converted to double-precision first, and then the operation is performed. Therefore, the result is a double-precision value.

   Example: 0#/3 (double-precision)

4. In logic operations, all numeric data is first converted into integer data. If any value cannot be converted into an integer within the range of –32768 to 32767, an error will occur.

   Example: LET A = NOT 12.34, –13 is assigned as A.

5. When a real number is converted into an integer, everything to the right of the decimal point is rounded off.

   Example: A = 12.3: “12” is assigned to A.

Expressions

Expressions refer to constants, variables, and functions that have been combined by operators. Numeric values, variables, or characters alone can also form expressions. There are four types of expressions:

- Arithmetic

- Relational
- Logical
- Character

Of these, the first three produce numeric values as a result, and are thus called, "numeric expressions." The last type is called a "character expression."

Arithmetic Operators

An arithmetic expression is made up of constants, variables, and functions combined using arithmetic operators. A list of valid arithmetic operators is shown in the following table.

Arithmetic Operator	Example	Operation
+	A + B	Addition
–	A – B, –A	Subtraction or negation
*	A * B	Multiplication
/	A / B	Real number division
\	A \ B	Integer division
MOD	A MOD B	Remainder after integer division
^	A ^ B	Exponentiation

Note If A or B is a real number in an expression using the \ or MOD operator, the decimal part is first rounded up to convert the real number into an integer, and then the operation is performed.

Relational Operators

Relational operators compare two values. The output is "–1" (&HFFFF) if the two values are equal and "0" if they are not.

Relational Operator	Example	Operation
=	A = B	Equal
<>,><	A <> B	Not equal
<	A < B	Less than
>	A > B	Greater than
≤	A ≤ B	Less than or equal to
≥	A ≥ B	Greater than or equal to

Character Operator

A character expression is made up of character constants and variables that are linked with the character operator "+". Instead of adding characters together, the "+" operator links the characters together to form one character value.

Input: A$="CF" B$="BASIC" PRINT A$+"–"+B$

Output: "CF–BASIC" is displayed.

Logical Operators

Logical Operators perform tests on multiple relations, bit manipulation, or boolean operations. The logical operator returns a bit result which is either "true" (not 0) or "false" (0). In an expression, logical operations are performed after arithmetic and relational operations. The outcome of a logical operation is determined

as shown in the following table. The operators are listed in the order of precedence.

Logical Operator	Description, Example, and Result	
NOT (negation)	A	NOT A
	1	0
	0	1
AND (logical product)	A B	A AND B
	1 1	1
	1 0	0
	0 1	0
	0 0	0
OR (logical sum)	A B	A OR B
	1 1	1
	1 0	1
	0 1	1
	0 0	0
XOR (exclusive-OR)	A B	A XOR B
	1 1	0
	1 0	1
	0 1	1
	0 0	0
IMP (implication)	A B	A IMP B
	1 1	1
	1 0	0
	0 1	1
	0 0	1
EQV (equivalence)	A B	A EQV B
	1 1	1
	1 0	0
	0 1	0
	0 0	1

Operator Priority

Arithmetic and logical operations are performed in the following order. Note, however, that an expression or function enclosed by parentheses is executed first, irrespective of operator priority.

1. ^ (exponentiation)
2. – (negation)
3. *, /
4. \
5. MOD
6. +. –
7. Relational operators
8. NOT
9. AND
10. OR
11. XOR
12. EQV
13. IMP

Calculation Examples of Logical Expressions

NOT (negation)

A =1= 0000000000000001

NOT 1 = 1111111111111110 = –2

NOT A = –2

AND (logical product)

A = 5 = 0000000000000101

B = 6 = 0000000000000110

A AND B = 0000000000000100 = 4

OR (logical sum)

A = 4 = 0000000000000100

B = 3 = 0000000000000011

A OR B = 0000000000000111 = 7

XOR (exclusive OR)

A = –4 = 1111111111111100

B = 5 = 0000000000000101

A XOR B = 1111111111111001 = –7

EQV (equivalent)

A = –4 =1111111111111100

B = 5 = 0000000000000101

A EQV B = 0000000000000110 = 6

IMP (implication)

A = –4 = 1111111111111100

B = 5 = 0000000000000101

A IMP B = 0000000000000111 = 7

4-2 Commands, Statements, and Functions

This section explains, in detail, the BASIC commands, statements and functions. They are presented in alphabetical order by section. Each description is formatted as described below.

4-2-1 BASIC Format

Purpose: Explains the purpose or use of the instruction

Format: Shows the correct format for the instruction

The following rules apply to the format descriptions of all commands, instructions, and functions:

- Items in CAPITAL LETTERS must be input as shown.
- Items in lower case letters enclosed in angle brackets (< >) are to be supplied by the user.
- Items in square brackets ([ ]) are optional.
- All punctuation marks except angle and square brackets (i.e., comas, hyphens, semicolons, parentheses, and equal signs) must be included where shown.
- Arguments to functions are always enclosed in parentheses. In the formats given for the functions in this chapter, the arguments have been abbreviated as follows:

x and y :	represent numeric expressions
I and J :	represent integer expressions
A$ and B$:	represent string expressions

Remarks: Explain in detail how to use the instruction

Examples: Show sample code to demonstrate the use of the instruction

4-2-2 Commands

This section describes all of the BASIC commands for the ASCII Unit.

AUTO Command

Purpose: To automatically generate line numbers for each line of the program

Format: AUTO [<line>][,[<increment>]]

<line> is a an integer from 0 to 63999.

<increment> is an integer value that specifies the increment of the generated line numbers.

Examples: AUTO 100, 10

AUTO 500, 100

Remarks:

Auto begins numbering at <line> and increments each subsequent line number by <increment>. The default value for both <line> and <increment> is 10.

The AUTO Command can be canceled by entering CTRL+C.

If an already existing line number is specified, an asterisk (*) is displayed immediately after the line number. If a new line number is input followed by a CR key, the new line number will be used instead. Pressing only the CR key leaves the line number unchanged.

CONT Command

Purpose: To resume execution of a program after a Ctrl+Break has been typed, a STOP or END statement has been executed, or an error has occurred

Format: CONT

Remarks:

Execution resumes at the point where the break occurred. If CTRL+X is pressed during data exchange with an external device, execution is aborted and the program cannot be resumed.

If the program is modified after execution has been stopped, the program can not be resumed.

CONT is usually used in conjunction with STOP for debugging.

DEL Command

Purpose: To Delete the specified program lines

Format: DEL [<first>] [-<last>] or DEL <first> -

<first> is the first line number deleted.

<last> is the last line number deleted.

Examples:

DEL 100	Deletes line 100
DEL 100-	Deletes all lines from line 100
DEL -150	Deletes all lines up to line 150
DEL 100-150	Deletes all lines between 100 and 150

Remarks:

A period may be used in place of the line number to indicate the current line.

EDIT Command

Purpose: To Edit one line of the program

Format: EDIT <line>

<line> is the line number to be edited.

Remarks:

The EDIT Command is used to display a specified line and to position the cursor at the beginning of that line. The cursor can then be moved within the specified line and characters can be inserted or deleted. Executing “EDIT .” will bring up the previously entered program line. “.” refers to the last line referenced by an EDIT statement, LIST statement, of error message.

LIST Command

Purpose: To list the program currently in memory on the screen or other specified device

Format: LIST [<line>] [-[<line>]]

LLIST [<line>] [-[<line>]]

<line> is a valid line number from 0 to 63339.

Remarks:

LIST displays a program or a range of lines on the screen or other specified device.

If the line range is omitted, the entire program is listed. “LIST.” displays or prints the line that was last input or was last displayed.

Output can be aborted by entering CTRL+B or CTRL+X. If CTRL+B is used, listing can be resumed by entering CTRL+B again.

LIST/LLIST Commands can be written into the program, but the following statement will not be executed and the ASCII Unit will enter command input wait status.

The LIST Command automatically outputs to port 1 and the LLIST Command automatically outputs to port 2.

The LLIST Command outputs data to the device “LPRT” independently of the OPEN statement.

When the dash (-) is used in a line range, three options are available:

1, 2, 3...

1. If only the first number is given, that line and all higher numbered lines are listed.
2. If only the second number is given, all lines from the beginning of the program through the given line are listed.
3. If both numbers are given, the inclusive range is listed.

Examples:

LIST -500	List everything up to line 500
LIST 10-100	List all lines ranging from 10 through 100
LIST 200-	List everything from line 200 on

LOAD Command

Purpose: To load a program from the EPROM into memory

Format: LOAD

Remarks:

The contents of the program area specified with the MSET Command are loaded from the EEPROM.

Purpose: To load a program sent from an RS-232C device to the current program area

Format: LOAD #<port>,“COMU:[<spec>,<vsl>]

<port> is either port 1 or port 2.

<spec>: see OPEN statement tables.

<vsl>: valid signal line—refer to the OPEN statement tables.

Example: LOAD #1,”COMU:(43)

Remarks:

When this command is executed, the BASIC indicator LED will begin blinking rapidly. Make sure the RS-232C device is connected at this time.

During execution of the LOAD command, the START/STOP switch and key input from port 1 will not be acknowledged.

The program area currently used is cleared immediately after the LOAD command is executed.

For details on communication parameters, valid signal lines, and COMU refer to the OPEN instruction.

MON Command

Purpose: To change to monitor mode

Format: MON

Remarks:

This Command passes control from BASIC mode to monitor mode.

To return to BASIC mode, enter CTRL+B.

In monitor mode, all Roman characters used must be in upper case.

MSET Command

Purpose: To reserve memory space for an assembly program

Format: MSET [<address>]

<address> is a hexadecimal number between &H200 and &H7FFF.

Example: MSET &H5000

Remarks:

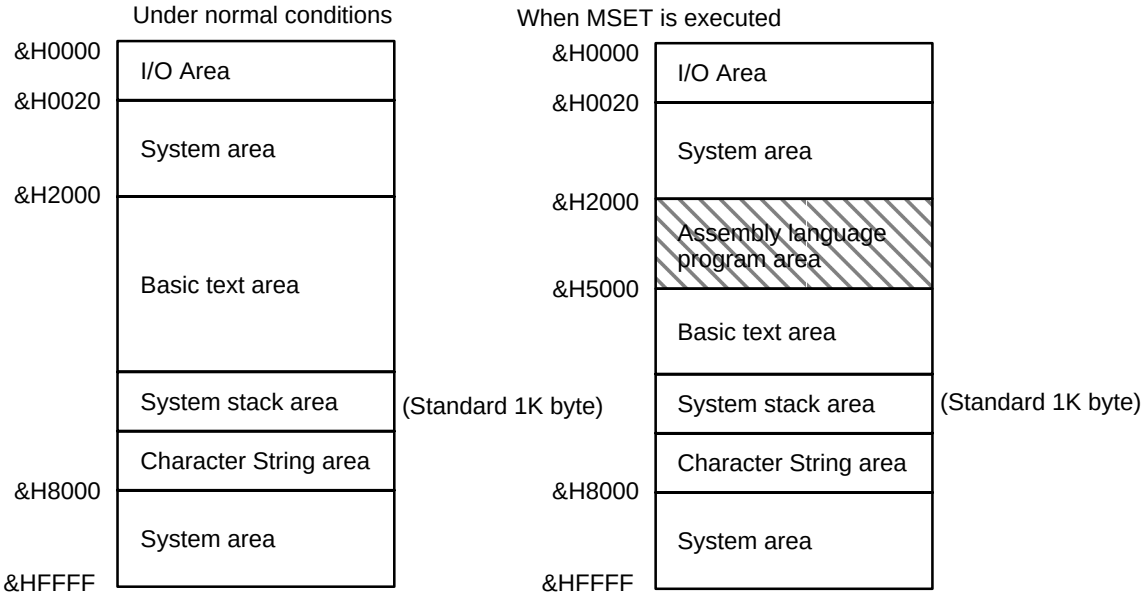
When an assembly program is to be used in conjunction with a BASIC program, special memory space must be reserved for the assembly program.

The MSET command sets the lowest possible address that a BASIC program can occupy. The assembly program is then stored “below” the BASIC program in memory. It is necessary to reserve enough space for the assembly program to “fit”.

If no MSET address is specified, the default MSET boundary address will be set at &H2000. Do not specify an address higher than &H7FFF or the system stack will be overwritten.

The address specified by this command is maintained even if system power is turned OFF. To cancel the effect of this command, execute MSET &H2000.

This diagram illustrates the PC memory map before and after the MSET command is executed.



NEW Command

Purpose: To delete the program currently in memory and clear all variables

Format: NEW

Remarks:

New is used to clear memory before a new program is entered. New causes all files and ports to be closed.

Programs named with the PNAME command cannot be erased. The name must therefore be erased first by executing PNAME " " before the NEW command is executed.

PGEN Command

Purpose: To select one of three program areas for the current program

Format: PGEN <num>

<num> is an integer of value 1, 2, or 3.

Remarks:

The occupied capacity of the selected program area will be displayed. (Refer to the discussion of the PINF command.)

PINF Command

Purpose: To display memory area information

Format: PINF [<arg>]

<arg> is either an integer of value 1, 2, or 3 or the character string "ALL". ALL is entered without quotes.

Examples: PINF 1

PINF ALL

Remarks:

This Command displays the amount of program area currently being used and the program names that have been assigned by the PNAME command. Specify 1, 2, or 3 as <arg> for a specific program area.

If <arg> is not specified, information on the area currently being used is displayed.

If ALL is specified, information on all three program areas will be displayed.

PNAME Command

Purpose: To assign a name to a program stored in the area specified with the PGEN command or to cancel a previously assigned program name

Format: PNAME <string>

<string> is the chosen name (enclosed in quotes) for the program or the null string, " ".

Examples: PNAME "PROG1"

PNAME " "

Remarks:

The chosen name must be eight characters or less.

Program areas assigned a name with the PNAME command are protected from execution of the LOAD and NEW commands which erase program area contents. It is necessary to erase all assigned program names with the PNAME " " command before execution of the LOAD or NEW commands.

RENUM Command

Purpose: To renumber program lines

Format: RENUM [<new number>] [,[<old number>][,<inc>]]

<new number> is the first line number to be used in the new sequence. The default is 10.

<old number> is the line in the current program where the renumbering is to begin. The default is the first line of the program.

<inc> is the increment to be used in the new sequence. The default is 10.

Examples: RENUM 200

RENUM 500, 200, 10

Remarks:

RENUM will also change all line number references following GOTO, GOSUB, THEN, ELSE, ON ... GOTO, ON ... GOSUB, RESTORE, RENAME, and ERL statements to reflect the new line numbers.

Statement numbers greater than 63999 cannot be used.

RUN Command

Purpose: To execute a program

Format: RUN [<line>]

<line> is any line number less than 63999.

Remarks:

If a line number is specified, execution begins from that line. If the line number is omitted, execution starts from the first line of the program.

The RUN command clears all variables and closes all open files before executing the designated program.

Program execution can be aborted with CTRL+X, or the START/STOP switch. Program execution can also be aborted from within the program by an END or STOP statement.

SAVE Command

Purpose: To write the program area to the EEPROM

Format: SAVE

Remarks:

The contents of the BASIC program area and the assembly language program area reserved with the MSET command are written to the EEPROM.

If the START/STOP switch is pressed during execution of the SAVE command, the process will be aborted.

Purpose: To write a program in the current program area to a storage device connected to one of the ports.

Format: SAVE #<port>,“COMU:[(<valid signal line>)]”

<port> is one of the two ports (1,2).

<valid signal line>: refer to the OPEN statement tables.

Example: SAVE #1,“COMU:(43)”

Remarks:

When this command is executed, the BASIC LED indicator on the ASCII Unit will blink rapidly warning the user to prepare the peripheral device for data transfer. When the device is set, press the START/STOP switch.

During execution of this command the START/STOP switch and key input through port 1 are inhibited.

For further details on COMU refer to the OPEN command.

TRON and TROFF Commands

Purpose: To trace execution of a program

Format: TRON

Remarks:

The TRON command is a debugging tool that enables the programmer to follow the execution of a program line by line. Execution of the TRON command will cause the line numbers of subsequent program statements to be displayed on the screen as they are executed.

The trace can be canceled with the TROFF command, the NEW command, by turning off the power or, with the RESET switch.

VERIFY Command

Purpose: To verify the contents of the EEPROM by comparing them to the contents of the program area

Format: VERIFY

Remarks:

If the contents of the program area are identical to those of the EEPROM, the message “READY” will be displayed; otherwise, the message “PROM ERROR” is displayed.

4-2-3 General Statements

CLEAR Statement

Purpose: To initialize numeric and character variables and set the size of the character memory area

Example: CLEAR [<size>]

<size> is the size of memory area used to process character strings and is specified in byte units.

Remarks:

This command initializes numeric variables to zero and character strings to empty. It also clears all user functions defined by the DEF FN statement.

This statement must be executed before the ON ERROR GOTO statement.

<size> is automatically set to 200 bytes upon power application or after reset.

COM Statement

Purpose: To enable, disable, or stop an interrupt defined by the ON COM GOSUB statement.

Format: COM[<port number>] ON/OFF/STOP

<port number> is an integer (1 or 2).

Example: COM1 ON

Remarks:

The COM ON statement enables an interrupt defined by the ON COM GOSUB statement.

After this statement has been executed, an interrupt will be generated each time data is written to the specified port buffer. The interrupt will cause program execution to branch to a routine defined by the associated ON COM GOSUB statement.

The COM OFF statement disables the com port interrupts. Even if data is written to a com port buffer, branching will not take place.

The COM STOP statement stops the com port interrupts from branching program execution. However, if the COM ON statement is subsequently executed, branching to the specified interrupt service routine based on the “STOPPED” interrupt will then take place.

If no port number is specified, port 1 is selected as the default port.

Execute the COM OFF statement at the end of the program.

The COM ON/OFF/STOP statement can be executed only after the ON COM GOSUB statement has been executed.

Program Example:

```
 10  OPEN #2, “COMU:”
 20  ON COM2 GOSUB 100
 30  COM2 ON
 40  GOTO 40
100  IF LOC(2)<>0 THEN A$=INPUT$ (LOC(2), #2)
110  RETURN
```

DATA Statement

Purpose: Defines numeric and character constants to be specified in a subsequent READ statement

Format: DATA <constant>[,<constant>]...

<constant> may be a numeric constant in any format; i.e., fixed-point, floating-point, or integer. <constant> can also be a character string. Quotes are only necessary if the constant contains comas, colons, or spaces.

Example: DATA CF, 10, 2.5, “A.:B”

Remarks:

Any number of DATA statements can be used in a program. READ statements access DATA statements in order (by line number). The data contained therein

may be thought of as one continuous list of items, regardless of how many items are on a line or where the lines are placed in the program.

DATA statements are non-executable and can be placed anywhere in a program. A data statement can contain as many constants as will fit on one line (separated by comas).

The variable type given in the READ statement must agree with the corresponding constant in the DATA statement.

DATA statements may be re-read from the beginning by use of the RESTORE statement.

No comment (with ":" or "'") can be written after the DATA statement.

DEF FN statement

Purpose: To define and name a function written by the user

Format: DEF FN<name>[(<arg1>[,<arg2>]...)] = <def>

<name>, which must be a legal variable name, is the name of the function.

<argn> is a list of variable names called parameters that will be replaced with values calculated when the function is called. The items in the list are separated by comas.

<def> is an expression that performs the operation of the function and is limited to one line.

Example: DEF FNA (X, Y, Z) = SQR(X**2 + Y**2 + Z**2)

Remarks:

A user function must be defined with the DEF FN statement before it can be called. To call a user function once it has been defined, append FN to the assigned name of the function and set it equal to some variable.

distance = **FN**A(X,5,5)

Variable names that appear in the defining expression serve only to define the function; they do not affect program variables that have the same name.

The variables in the parameter list represent, on a one-to-one basis, the argument variables or values that will be given in the function call.

This statement may define either numeric or string functions. If a type is specified in the function name, the value of the expression is forced to that type before it is returned to the calling statement.

If a type is specified in the function name and the argument type does not match, an error will occur.

DEF INT/SNG/DBL/STR Statement

Purpose: To declare variable types as integer, single-precision, double-precision, or string

Format: DEF <type><letter>[-<letter>]

[<letter>[-<letter>]]...

<type> is INT, SNG, DBL, or STR

Remarks:

Any variable names beginning with the <letter(s)> listed will automatically be assigned to the specified variable type.

The "%", "!", and "$" declaration characters take precedence over a DEF <type> statement.

If no type declaration statements are encountered, BASIC assumes all variables without declaration characters to be single-precision variables.

Example: DEFINT A-D, X

All variables beginning with A, B, C, D, and X will be integer variables.

DEF USER Statement

Purpose: To specify the starting address of an assembly language subroutine that will be called via the USR function

Format: DEF USR [<digit>] = <offset>

<digit> is an integer from 0 to 9. The digit corresponds to the USR routine number whose address is being specified. If <digit> is omitted, DEF USR0 is assumed.

<offset> is the starting address of the USR routine.

Remarks:

Any number of DEF USR statements may appear in a program to redefine subroutine starting addresses, thus allowing access to as many subroutines as necessary.

Program Example:

```
100   DEF USR1=&H2100
110   POKE &H2100, &H39
120   A=USR1 (A)
130   PRINT A
```

DIM Statement

Purpose: To specify the maximum values for array variable subscripts and allocate storage accordingly

Format: DIM <variable>(<subscripts>)

[,<variable>(<subscripts>)]...

<variable> is a legal variable name.

<subscripts> are the maximum number of elements for each dimension of the array. There can be up to 255 subscripts but the maximum size of the array cannot exceed the amount of memory available.

Example: DIM A (10,20), B$(30)

Remarks:

If an array variable name is used without a DIM statement, the maximum value of the array's subscript(s) is assumed to be 10. If a subscript is used that is greater than the maximum specified, an error will occur. The minimum value for a subscript is zero.

The DIM statement initializes all the elements of numeric arrays to zero. String array elements are initialized to NULL.

END Statement

Purpose: To terminate program execution, close all files, and return to command level

Format: END

Remarks:

END statements may be placed anywhere in the program to terminate execution. Unlike the STOP statement, END closes all open files or devices. An END statement at the end of the program is optional. BASIC always returns to command level after an END is executed.

ERROR Statement

Purpose: To simulate the occurrence of an error, or to allow error codes to be defined by the user

Format: ERROR <n>

<n> is the error code to be simulated.

Remarks:

Error code numbers 1 to 255 are predefined and reserved by BASIC. Higher numbers can be used for user-defined error code messages. User-defined error codes can be used together with the ON ERROR GOTO statement to branch the program to an error handling routine.

When the ERROR statement is executed without an accompanying ON ERROR GOTO statement, the error message corresponding to the specified error number is output and program execution is stopped. The message UNDEFINED ERROR is displayed if an undefined error occurs.

The error number is assigned to the variable ERR and the line number where the error occurred is assigned to the variable ERL.

FOR and NEXT Statements

Purpose: To allow a series of instructions to be performed in a loop a given number of times

Format: For <var>=<x> TO <y> [STEP<z>]

<x>, <y>, and <z> are numeric expressions.

Example: 100 FOR Y = base TO 10 STEP 2

110 NEXT Y

Remarks:

<var> is used as a counter. The first numeric expression (<x>) is the initial value of the counter. The second numeric expression (<y>) is the final value of the counter.

The program lines following the FOR statement are executed until the NEXT statement is encountered. Then the counter is incremented by the amount specified by STEP.

A check is performed to see if the value of the counter is now greater than the final value (<y>). If it is not greater, execution branches back to the first statement after the FOR statement and the process is repeated. If it is greater, execution continues with the statement following the NEXT statement. This is a FOR...NEXT loop.

If STEP is not specified, the increment is assumed to be one. If STEP is negative, the counter will count down instead of up. In this case, the loop will be executed until the counter is less than the final value.

The body of the loop will never be executed if the initial value of the loop is greater than the final value.

NESTED LOOPS

FOR...NEXT loops may be nested, that is, a loop can be placed inside of another loop. When loops are nested, each loop must have a unique variable name for its counter. The NEXT statement for the inside loop must come before the NEXT statement for the outer loop.

If nested loops have the same end point, the same NEXT statement can be used for both of them.

If a NEXT statement is encountered before its corresponding FOR statement, an error message is issued and execution is terminated.

GOSUB and RETURN Statements

Purpose: To branch to and return from a subroutine

Format: GOSUB <line>

<line> is the first line number of the subroutine.

Remarks:

A subroutine may be called any number of times in a program, and a subroutine may be called from within another subroutine.

The RETURN statement(s) in a subroutine causes execution to branch back to the statement following the most recent GOSUB statement.

A subroutine may contain more than one RETURN statement should logic dictate a return at different points in the subroutine.

Subroutines can appear anywhere in the program, but it is recommended that subroutines be readily distinguishable from the main program.

To prevent inadvertent entry into a subroutine, the subroutine may be preceded by a STOP, END, or GOTO statement to direct program execution around the subroutine.

Program Example:

```
 10    T = Time
 20    GOSUB 100
 30    {stuff}
 40    .
 50    .
 60    .
 90    GOTO 150
100
110    T = T + TIME
120    RETURN
130    {stuff}
```

GOTO Statement

Purpose: To unconditionally branch program execution to the specified line number

Format: GOTO <line>

<line> is a valid line number.

Remarks:

If <line> is a non-executable statement, execution will proceed at the first executable statement encountered after <line>.

IF...THEN Statements

Purpose: To control program flow based on the results returned by an arithmetic or logical expression

Format: IF <expression> [,] THEN <statement(s)> or <line>

[ELSE <statement(s)> or <line>]

IF <expression> [,] GOTO <line>

[[,] ELSE <statement(s)> or <line>]

Example: IF B=10 THEN PRINT “hello” ELSE 500

Remarks:

If the result of <expression> is not zero, the THEN or GOTO clause will be executed (GOTO is always followed by a line number). THEN may be followed by either a line number for branching or one or more statements to be executed.

If the result of <expression> is zero, the THEN or GOTO clause will be ignored and the ELSE clause, if present, will be executed. IF there is no ELSE clause, execution will continue with the next executable statement.

INPUT Statement

Purpose: To allow input from the keyboard during program execution

Format: INPUT [;] [#<port>][<“prompt”>;]<variable>

[,<variable>]...

#<port> is the port number (1 or 2).

<“prompt”> is a message that will be displayed when the INPUT statement is executed.

Examples: INPUT “DATA” : A$

INPUT #2, “DATA” , A$, B$

Remarks:

When an INPUT statement is executed, program execution pauses and a question mark is displayed to indicate the program is waiting for data. If <“prompt”> is included, the string is displayed before the question mark. The program will not continue execution until the user has entered the required data.

A coma may be used instead of a semicolon after the prompt string to suppress the question mark.

Data is not excepted by the INPUT statement until a carriage return is entered. Therefore input can be edited with the backspace and delete keys.

When more than two variables are input, they must be delimited by a coma(s).

The data entered is assigned to the variables specified by the INPUT statement. The number of values entered must be the same as the number of variables in the INPUT statement.

The variable names in the list may be numeric or string variable types as well as subscripted variables (array variable). The type of each entered data item must agree with the type specified by the variable name.

Strings input to an INPUT statement need not be surrounded by quotation marks.

Responding to INPUT with too many or too few items will cause an error message to be displayed prompting the user to re-enter the data.

If a peripheral device other than TERM or COMU is selected by the OPEN statement, neither the prompt statement nor “?” is displayed.

To eliminate “?” when COMU, etc., is selected by the OPEN statement, use the LINE INPUT command.

The INPUT statement cannot be executed in direct mode. If the port number is omitted, port 1 is assumed as the default port.

KEY(n) Statement

Purpose: To enable, disable, or stop an interrupt invoked by key input and defined by the ON KEY GOTO or ON KEY GOSUB statements

Format: KEY(<n>) ON/OFF/STOP

<n> is the key number (1-8).

Example: KEY(4) ON

Remarks:

The KEY ON statement enables an interrupt invoked by keyboard input. After this statement has been executed, an interrupt will be triggered each time the specified key is input. Program execution then branches to an interrupt service routine defined with the ON KEY GOTO or ON KEY GOSUB statements.

The KEY OFF statement disables the interrupt; key input will no longer trigger an interrupt.

The KEY STOP statement also disables the interrupt. However, if the interrupt is subsequently enabled with the KEY ON statement, execution will then branch to the interrupt service routine defined by the ON KEY GOTO or ON KEY GOSUB statements.

Execute the KEY OFF statement at the end of the program.

Program Example:

```
 10    OPEN #1, “TERM:(42)”
 20    ON KEY 1 GOSUB 100
 30    On KEY 2 GOSUB 200
 40    A=0
 50    KEY ON
 60    GOTO 60
100    PC READ “14”;A
110    RETURN
200    PC WRITE “14”;A
210    RETURN
```

LET Statement

Purpose: To assign the value of an expression on the right side of an equal sign to the variable on the left side

Format: [LET] <variable>=<expression>

Example: LET A = 1.2

Remarks:

Notice the word LET is optional, i.e., the equal sign is sufficient when assigning an expression to a variable name.

Assignment of a character variable to a numeric variable, and the reverse, are not permitted.

When assigning unmatched types of numeric variables, the variable type on the right side of the equal sign is converted into the type on the left before the assignment is performed.

String assignments should be enclosed in double quotation marks.

LINE INPUT Statement

Purpose: To input an entire line of characters (up to 255) from the keyboard or other input device without the use of delimiters

Format: LINE INPUT [#<port>,] [“<prompt>”;]<string>

<port> is the port number (1 or 2).

“<prompt>” is a message displayed on the screen prompting the user for input.

<string> is a string variable that is assigned to the input character string.

Example: LINE INPUT #2,”DATE”;A$

Remarks:

All of the characters input from the end of the prompt to the carriage return are assigned to the character variable as a series of data. (Comas and colons are also treated as character data.)

A question mark is not displayed unless it is part of the prompt string.

The prompt statement is not displayed if a peripheral device other than TERM or COMU is selected with the OPEN statement.

The character string is not assigned to the variable until the carriage return key is pressed. Until then, the BASIC LED indicator on the ASCII Unit will blink indicating that the Unit is waiting for input of a carriage return.

If the port number is omitted, port 1 is assumed as the default port.

MID$ Statement

Purpose: To replace a portion of one string with another string

Format: MID$(<string 1>,<n>[,<m>]) = <string 2>

<string 1> is a string variable.

<n> is an integer expression from 1 to 255.

<m> is an integer expression from 0 to 255.

<string 2> is a string expression.

Example: MID$(A$,2,4) = “ABCDEFGH”

Remarks:

The characters in <string 1>, beginning at position <n> are replaced by the characters in <string 2>.

The optional <m> refers to the number of characters from <string 2> that will be used in the replacement. If <m> is omitted, all of <string 2> is used. However, regardless of whether <m> is included or not, the replacement of characters never goes beyond the original length of <string 1>.

Refer to the discussion of the MID$ function

ON COM GOSUB Statement

Purpose: Defines an interrupt service routine to handle data coming into a com port buffer

Format: ON COM(<n>) GOSUB <line>

<n> is the port number (1 or 2).

<line> is the line number of the first statement of the interrupt service routine.

Example: ON COM1 GOSUB 1000

Remarks:

This statement is not valid unless it is executed after the specified port has been opened.

An interrupt service routine cannot be interrupted by another interrupt. If a new interrupt occurs during processing of a previous interrupt, branching to handle the new interrupt will not take place until after the RETURN statement of the first interrupt service routine is executed. This means that, depending on the branch timing, nothing may be in the buffer when execution branches to the interrupt routine. It is therefore necessary to check whether data is in the buffer by executing the LOC or EOF Command at the beginning of the interrupt routine.

All subroutines must end with a RETURN statement.

If a statement specified by the branch line number is non-executable, execution will begin with the first executable statement following the branch line number.

If zero is specified as the branch line number, it is assumed that the COM OFF statement has been executed.

If the port number is omitted, port 1 is selected.

The ON COM GOTO statement is enabled with the COM ON statement and disabled with the COM OFF statement.

Program Example:

```
 10    OPEN #1, “COMU:(40)”
 20    ON COM GOSUB 100
 30    COM ON
 40    PC READ “2I4”;A,B
 50    PRINT A, B
 60    GOTO 30
100    IF LOC (1)=0 THEN 120
110    PRINT INPUT$ (LOC(1),#1)
120    RETURN
```

Program Remarks:

If an interrupt from port 1 is detected, the buffer contents are displayed.

ON ERROR Statement

Purpose: To enable error processing and to specify the first line number of the error handling routine

Format: ON ERROR GOTO <line>

<line> is any valid line number.

Remarks:

When an error occurs, this statement directs execution to the proper error handling routine. When an error is detected, the error number is assigned to the variable ERR and the line number where the error occurred is assigned to ERL.

To disable error processing, execute ON ERROR GOTO 0. Subsequent errors will cause an error message to be printed and execution to be halted.

If an error occurs during execution of an error handling subroutine, a BASIC error message will be printed and execution terminated.

Refer to the discussion of the RESUME Command, and the ERR and ERL functions.

ON GOSUB and ON GOTO Statements

Purpose: To branch to one of several specified line numbers, depending on the resultant evaluation of a numeric or logical expression

Format: ON <expression> GOTO <list>

ON <expression> GOSUB <list>

<expression> is any valid expression.

<list> is a list of valid line numbers separated by comas.

Example: ON X–2 GOSUB 50,100,150

Remarks:

The value of <expression> determines which line number in the list will be used for branching. For example, if the result is 2, then the second line number in the list will be chosen for branching. If the resultant value is not an integer, the fractional part is rounded off.

In the ON...GOSUB statement, each line number in the list must be the first line number of a subroutine.

If the value of <expression> is zero or greater than the number of items in the list, execution continues with the next executable statement. If the value of <expression> is negative or greater than 255, an error message will be displayed.

ON KEY GOSUB Statement

Purpose: Defines an interrupt service subroutine to handle specific keyboard input

Format: ON KEY(<n>) GOSUB <line>

<n> is a numeric expression from one to eight indicating a specific key.

Example: ON KEY 1 GOSUB 1000

Remarks:

An interrupt service routine cannot be interrupted by another interrupt. If a new interrupt occurs during processing of a previous interrupt, branching to handle the new interrupt will not take place until after the RETURN statement of the first interrupt service routine is executed.

If a statement specified by the branch line number is non-executable, execution will begin with the first executable statement following the branch line number.

If zero is specified as the branch line number, it is assumed that the KEY OFF statement has been executed.

If the port number is omitted, port 1 is selected.

There should be only one ON KEY GOTO statement for each key number.

Key input will not be processed during execution of an assembly language program.

The ON KEY GOSUB statement is enabled with the KEY ON statement and disabled with the KEY OFF statement.

Program Example:

```
 10  OPEN #1,“TERM:(42)”
 20  ON KEY 1 GOSUB 100
 30  ON KEY 2 GOSUB 200
 40  ON KEY 3 GOSUB 300
 50  KEY ON
100  PRINT A
110  RETURN
200  PRINT B
210  RETURN
300  PRINT C
310  RETURN
```

Program Remarks:

“A”, “B”, and “C” are displayed by pressing keys 1, 2, and 3, respectively. To cancel the specification, write 0 as the branch destination.

ON KEY GOTO Statement

Purpose: To branch program execution to a specified line number in response to a specific key input

Format: ON KEY<n> GOTO <line>

<n> is an integer in the range of 1 to 8.

<line> is any valid line number.

Example: ON KEY 1 GOTO 1000

Remarks:

If a statement specified by the branch line number is non-executable, execution will begin with the first executable statement following the branch line number.

If zero is specified as the branch line number, it is assumed that the KEY OFF statement has been executed.

If the port number is omitted, port 1 is selected.

There should be only one ON KEY GOTO statement for each key number.

Key input will not be processed during execution of an assembly language program.

The ON KEY GOTO statement is enabled with the KEY ON statement and disabled with the KEY OFF statement.

Program Example:

```
 10  OPEN #1,“TERM:(42)”
 20  ON KEY 1 GOTO 100
 30  ON KEY 2 GOTO 200
 40  ON KEY 3 GOTO 300
 50  KEY ON
100  PRINT “A”
110  GOTO 500
200  PRINT “B”
210  GOTO 5000
300  PRINT “C”
500  {cont. processing}
```

Program Remarks:

“A”, “B”, and “C” are displayed by pressing keys 1, 2, and 3, respectively. To cancel the specification, write 0 as the branch destination.

ON PC ... GOSUB Statement

Purpose: Defines an interrupt service routine invoked by the PC

Format: ON PC [<int num>] GOSUB <line>

<int num> is an integer from 1 to 15.

<line> is a valid line number.

Example: ON PC 3 GOSUB 1000

Remarks:

In four-word mode, the interrupt source number is indicated with bits 04 to 07 (1 to F in hexadecimal) of word n+1. In two-word mode, the interrupt source number is indicated with bits 00 to 07 of word n.

An interrupt routine invoked by the ON PC statement cannot be interrupted by another interrupt. If a new interrupt occurs during processing of a previous interrupt, branching to handle the new interrupt will not take place until after the RETURN statement of the first interrupt service routine is executed.

If the statement specified by the branch line number is non-executable, execution will begin with the first executable statement following the branch line number.

If zero is specified as the branch line number, it is assumed that the KEY OFF statement has been executed.

If the interrupt number is omitted, the same branch destination is assumed for all interrupt numbers, 1 to 15.

The ON PC GOSUB statement is enabled with the PC ON statement and disabled with the PC OFF statement.

Program Example:

```
 10    ON PC 1 GOSUB 100
 20    ON PC 2 GOSUB 200
 30    PC ON
 40    GOTO 40
100    PC READ “H4,I2”;I, J
110    PRINT I, J
120    RETURN
200    INPUT A
210    PC READ “H4,I2”;K,L
220    PC WRITE “14”; A
230    RETURN
```

Program Remarks:

When interrupt 1 is invoked, program execution branches to statement 100, reads two words of data from the PC, and displays them on the CRT.

When interrupt 2 is invoked, program execution branches to statement 200 and writes data entered through the keyboard to the PC.

PC GET Statement

Purpose: To read output data from the PC

Format: PC GET <var 1>[,<var 2>]

Example: PC GET I,J

Remarks:

In two-word mode, bits 0 to 7 of word (n) are read and assigned to <var 1>. Bits 8 to 15 of data word (n) are read and assigned to <var 2>. In four-word mode, the same bits are assigned from word (n+1).

The ASCII Unit converts the hexadecimal data into decimal data (0 to 255) before assigning it to the specified variables.

PC ... ON/STOP Statements

Purpose: To enable or stop a PC interrupt defined with an ON PC GOSUB statement

Format: PC [<num>] ON/STOP

<num> is a specific interrupt number.

Remarks:

The PC ON statement enables an interrupt defined by the ON PC GOSUB statement.

After this statement has been executed, each PC interrupt will cause program execution to branch to a routine defined by the associated ON PC GOSUB statement.

The PC STOP statement disables PC interrupts from branching program execution. However, if the PC ON statement is subsequently executed, execution will branch to the specified interrupt service routine based on the “STOPPED” interrupt.

The PC ON/STOP statements can be executed only after the ON PC GOSUB statement has been executed.

If there is more than one interrupt routine in the program the specific interrupt number should be specified. If there are two or more routines and the interrupt

number is not specified, the routine closest to the end of the program or at the highest line number will be executed regardless of which interrupt is invoked.

After the ON PC GOSUB statement is executed, PC ON becomes valid. Refer to the following example.

Program Example:

```
 10    ON PC GOSUB 100
 20    PC ON
 30    GOTO 30
100    PC READ “3I2”; A, B, C
110    PRINT A, B, C
120    RETURN
```

PC PUT Statement

Purpose: To write data to the PC’s ASCII Unit Data Memory Area

Format: PC PUT <num exp>

<num exp> is a valid numeric expression between 0 and 255.

Examples: PC PUT I

PC PUT 123

Remarks:

In two-word mode, data is written to bits 8 to 15 of word n+1. In four-word mode, data is written to bits 8 to 15 of word n+3.

If the value of the numeric expression is not an integer, the INT function is internally executed to round it off. If the value of the numeric expression is negative or greater than 255, zero is written to the PC.

PC READ Statement

Purpose: To read data from the PC

Format: PC READ “<format>[,<format>,<format>, ...]”;
<var1>[,<var2>,]...

<format> specifies how the data will be read. For specific format information, refer to Appendix C.

Examples:

PC READ “2H1, A3, I4, O2”; X, Y, A$, I, J

Remarks:

When the PC has written the data to the ASCII Unit, the PC READ statement is executed.

If the PC has not written the data to the ASCII Unit, the ASCII Unit will wait for the data, and the PC READ statement is not executed until the data comes.

If the number of data items output by the PC is greater than that specified by the format parameters, the excess part of the output data will be ignored.

The maximum number of data items that can be transferred with one READ statement specification is 255 in the S or A formats.

If an amount of memory greater than the actual memory area is specified by the READ statement, a FORMAT ERROR will occur.

The PC READ statement’s formatting parameters can be assigned to a single character variable and that variable may then be used in the PC READ statement.

Refer to Appendix C for details on READ and WRITE statement formatting.

Example:

```
A$ = “2H1, A3, I4, O2”
PC READ A$;X, Y, A$, I, J
```

PC WRITE Statement

Purpose: To write data to the PC

Format: PC WRITE “<format>[,<format> ...]”;<exp1> [,<exp2>, ...]

Note For parameter definitions, refer to the PC READ instruction.

Examples:

PC WRITE “H4, A2, I3, O4”; 1234, “AB”, K, L

Remarks:

If the data of the previous PC WRITE statement has not been read by the PC, the next PC WRITE statement cannot be executed until the previous one is completed.

The maximum number of data items that can be transferred with one WRITE statement specification is 255 in the S or A formats.

If an amount of memory greater than the actual memory area is specified by the WRITE instruction, a FORMAT ERROR will occur.

If the value of <exp> is not an integer, the INF function is internally executed to round it off.

Single-precision and double-precision numeric expressions are internally converted into integer expressions.

The PC WRITE statement’s formatting parameters can be assigned to a single character variable and that variable may then be used in the PC WRITE statement.

Example:

A$=“H4, A2, I3, O4”

PC WRITE A$; 1234, “AB”, K, L

POKE Statement

Purpose: To write one byte to a specified memory address

Format: POKE <address>,<data>

<address> is the memory location where data will be POKEd.

<data> is an integer from 0 to 255.

Example: POKE &H2000,&H39

Remarks:

The address must be a 2-byte integer ranging from 0 to 65535 (&HFFFF). Do not write data to addresses &H0000 to &H1FFF, and &H8000 to &HFFFF; they are reserved for system use.

PRINT Statement

Purpose: To output data and text to the screen or printer

Format: PRINT [#<port>,] [<list of exp>][;]

LPRINT

<port> is an integer (1 or 2).

<list of exp> can be numeric or character expressions. Character expressions should be enclosed in double quotation marks.

Example: PRINT #1,A,B$;“BASIC”

Remarks:

The list of expressions must be separated by comas, semicolons, or blanks. When the expressions are separated with blanks or semicolons, the next value

is output immediately after the preceding value. When the expressions are separated with comas, the values are output at intervals of nine characters.

If the list of expressions is not terminated with a semicolon, a carriage return is appended after the last expression.

If numeric expressions are used, a blank is output before and after the resultant value. The blank before the value is used for a minus sign, if one is required.

If <list of exp> is omitted, execution of this statement causes a carriage return to be output.

If the port specification is omitted, port 1 is assumed for the PRINT statement, and port 2 for the LPRINT statement.

The LPRINT statement outputs data under control of the device connected to port 2, irrespective of the OPEN statement directives.

LPRINT USING Statement

Purpose: To output strings or numbers according to a specified format

Format: PRINT [#<port>,] USING “<format>”; <list of exp>

Example: PRINT #1, USING “####,# \\###”;A;B

Remarks:

The following characters control the format of the output:

!	Outputs the first character only.
& &	Outputs the characters enclosed by &.
@	Outputs the corresponding character string.
#	Outputs the corresponding character string.
.	Inserts a decimal point at any desired place.
+	Places a plus sign before and after a numeric value.
-	Places a minus sign before and after a numeric value. (Write this character at the end of the format character string.)
**	Places two asterisks in the blank, upper digit positions of a numeric value.
\\	Places one \ in the blank digit position immediately before a numeric value.
**\	Combines the functions of ** and \\.
’	Delimits an integer at every third digit position from the right.
^^^^	Indicates the output in exponential format (E+nn). Add this character after #.
””	is output before the numeric value if the specified number of digits is too great.

If the port number is omitted, port 1 is assumed for the PRINT USING statement and port 2 for the LPRINT USING statement.

The LPRINT statement outputs data under control of the peripheral device connected to port 2 irrespective of the OPEN statement directives.

RANDOM Statement

Purpose: To reseed the random number generator

Format: RANDOM [<exp>]

<exp> is a single or double-precision integer that is used as the random number seed.

Example: RANDOM 5649

Remarks:

The value of <exp> should be from -32768 to 32767. If the expression is omitted, a message requesting the random number seed will be displayed.

If the random number generator is not reseeded, the RND function returns the same sequence of random numbers each time the program is run. To change the sequence of random numbers each time the program is RUN, place a RANDOM statement at the beginning of the program and change the seed with each RUN.

For more information, refer to the explanation of RND.

READ Statement

Purpose: To read values from a DATA statement and assign them to the specified variables

Format: READ <list of var>

Example: READ A,B$

Remarks:

A read statement must always be used in conjunction with a DATA statement. READ statements assign variables to DATA statement values on a one-to-one basis. READ statement variables may be numeric or string, and the values read must be the same type as the corresponding variable. If they do not agree, a syntax error will occur.

A single READ statement may access one or more DATA statements (they will be accessed in order), or several READ statements may access the same DATA statement.

If the number of variables in <list of var> exceeds the number of elements in the DATA statement(s), an error message will be displayed. If the number of variables specified is fewer than the number of elements in the DATA statement(s), subsequent READ statements will begin reading data at the first unread element. If there are no subsequent READ statements, the extra data is ignored.

To reread DATA statements from the beginning, use the RESTORE statement.

REM Statement

Purpose: To insert non-executable comments in a program

Format: REM <remark>

<remark> text does not need to be enclosed in quotes.

Example: REM SAMPLE PROGRAM

Remarks:

The REM statement is used to provide titles to programs and to insert helpful comments to be used during program debugging or modification.

Remarks may be added to the end of a line by preceding the remark with a single quotation mark instead of REM.

Do not use a REM statement in a DATA statement as it will be taken as legal data.

RESTORE Statement

Purpose: To allow DATA statements to be reread from a specified line

Format: RESTORE [<line>]

<line> should be the line number of a valid DATA statement.

Example: RESTORE 1000

Remarks:

This statement causes the next READ statement to read the first element in the first DATA statement that exists in the program. If <line> is specified, the next READ statement accesses the first item in the specified DATA statement.

RESUME Statement

Purpose: To resume program execution after an error handling procedure has been performed

Formats: RESUME [0]: execution resumes at the statement which caused the error.

RESUME NEXT: execution resumes at the statement immediately following the one which caused the error.

RESUME <line>: execution resumes at <line>.

Example: RESUME 100

Remarks: Any one of the above formats may be used.

STOP Statement

Purpose: To terminate program execution and return to the BASIC command level

Format: STOP

Remarks:

Execution of this statement causes the message “BREAK IN xxxx” to be displayed and the ASCII Unit to return to the command level.

The ports will not be closed.

Program execution can be resumed with the CONT command.

WAIT Statement

Purpose: Sets a time limit for the execution of a specific statement

Format: WAIT “<wait time>”[,<line number>]

<wait time> is the allowable time for the monitored statement to be executed.

<line number> is any valid line number.

Example: WAIT “10:30.5”,100

Remarks:

The delay time is set in the form MM.SS.F, where:

MM is the number of minutes - up to 59

SS is the number of seconds

F is tenths of seconds.

The statement immediately following the WAIT statement is the monitored statement. If execution of this statement is not completed within the set wait time, program execution will branch to <line number>.

Interrupts invoked by the ON COM, ON KEY, ON PC, or ON ERROR statements will not be recognized until after the WAIT statement or the monitored statement has been processed.

The WAIT statement can monitor the following statements:

INPUT, INPUT$, LINE INPUT, PC READ, PC WRITE, PRINT, LPRINT, PRINT USING, LPRINT USING

If a statement other than one of those listed above is specified to be monitored by a WAIT statement, and if execution of that statement is not completed within the set time of the WAIT statement, an error will occur.

Program Example:

```
 10    WAIT “10.0”, 100
 20    PC READ “3I4”; A, B, C,
 30    PRINT A, B, C
 40    END
100    PRINT “PC ERR”
110    GOTO 40
```

Program Remarks:

This example will display the message “PC ERR” if the PC READ statement is not executed within 10 seconds.

4-2-4 Device Control Statements

This section describes statements that control hardware and communications.

CLOSE Statement

Purpose: To close a port

Format: CLOSE [#<port>]

<port> is an integer (1 or 2).

Remarks:

If the port number is omitted, both ports will be closed.

Once the port has been closed, it cannot be used for data transfer until it is opened again.

Be sure to execute the CLOSE statement to correctly end the output process. CLOSE dumps any data remaining in the buffer from output operations. It does not dump data from input operations.

To turn OFF the error indicators at Port 1 and Port 2 or error bits that are ON due to a transmission error or reception buffer overflow, execute the CLOSE statement.

The END statement and the NEW command automatically close the ports, but the STOP statement does not.

CLS Statement

Purpose: To clear the screen

Format: CLS [#<port>]

<port> is an integer (1 or 2).

Remarks:

This statement clears the screen and moves the cursor to the home position. If the port number is omitted, port 1 is assumed.

OPEN Statement

Purpose: To allow input/output operations to take place through the specified port

Format: OPEN #<port>, “<device name>:[(<com spec. or vsl>)]”

<port> is an integer (1 or 2).

<device name> identifies the device.

<com spec> stands for the communication specifications.

<vsl> stands for valid signal line.

Examples: OPEN #1,“KYBD:”

OPEN #2,“COMU:(14)”

The following three tables define the communication parameters for the OPEN Statement.

Peripheral Device	Name	Output from ASCII Unit	Input to ASCII Unit
Terminal	TERM:	YES	YES
Keyboard	KYBD:	NO	YES
Display	SCRN:	YES	NO
Printer	LPRT:	YES	NO
RS-232C device	COMU:	YES	YES

Note TERM cannot be used with port 2.

Communication Specifications	Character Length	Parity	Stop Bit
0	7 bits	Even	2 bits
1	7 bits	Odd	2 bits
2	7 bits	Even	1 bit
3	7 bits	Odd	1 bit
4	8 bits	None	2 bits
5	8 bits	None	1 bit
6	8 bits	Even	1 bit
7	8 bits	Odd	1 bit

Signal Line	CTS	DSR	RTS	XON / XOFF
0	Valid	Valid	Valid	Invalid
1	Valid	Valid	Invalid	
2	Valid	Invalid	Valid	
3	Valid	Invalid	Invalid	
4	Invalid	Valid	Valid	
5	Invalid	Valid	Invalid	
6	Invalid	Invalid	Valid	
7	Invalid	Invalid	Invalid	
8	Valid	Valid	Valid	Valid
9	Valid	Valid	Invalid	
A	Valid	Invalid	Valid	
B	Valid	Invalid	Invalid	
C	Invalid	Valid	Valid	
D	Invalid	Valid	Invalid	
E	Invalid	Invalid	Valid	
F	Invalid	Invalid	Invalid	

Note To make the CTS signal invalid at port 2, pull the CTS line high or connect it to the RTS line.

When the RTS is specified to be ON (valid), the RTS signal goes high when the port is opened and remains high until the port is closed. When the RTS signal is specified to be OFF (invalid), the RTS signal remains low unless an I/O statement such as PRINT or INPUT is executed.

If XON is designated, the XOFF code will be transmitted and the ASCII Unit will request the interruption of transmission when the buffer is 3/4 full at the time of data reception. The XON code will be transmitted and the ASCII Unit will request the restart of transmission if the buffer becomes 1/4 full. Data transmission will be interrupted if the XOFF code is received and data transmission will restart when the XON code is received. If XOFF is designated, control is not possible. This means, if the buffer is full, no more data can be received.

If the communication specification and the valid signal line are omitted, their defaults are:

Peripheral Device	Communication Conditions	Valid Signal Line
Terminal	4	3
Keyboard	4	3
Display	4	3
RS-232C device	4	3
Printer	4	3

Ports already open cannot be opened again. When the OPEN and CLOSE statements are used, port 1 is assumed to be for a terminal and port 2 is assumed to be for a printer. Port 2 cannot be selected for a terminal.

I/O statements specifying #<port> cannot be used to transfer data through a port that has not been opened with the OPEN statement. To input/output data in the case where the OPEN statement has not been executed, use the I/O statements without the #<port> specification.

The following two tables illustrate peripheral device output levels during execution of the OPEN statement.

Device	When Opened		During Operation	
	RTS	DTR	RTS	DTR
TERM	LOW	HIGH	HIGH	No change
SCRN	LOW	LOW	HIGH	No change
KEYB	LOW	HIGH	HIGH	No change
COMU	LOW	HIGH	HIGH	No change
LPRT	LOW	LOW	HIGH	No change

Device	When Closed	
	RTS	DTR
1	LOW	HIGH
2	LOW	LOW

Note The default selection for the ports is as follows:

Port 1: Terminal device

Port 2: Printer

The following table presents the output control codes for the terminal, printer, and COMU device.

SCRN TERM	Clears the screen buffer when code &H0C (CLR) is output.The column position is set to 0 (i.e., the leftmost position) when code &H0A (LF), &H0D (CR), &H0B (HOME), or &H08 (BS) is output. The cursor is moved as specified on the screen when code &H08 (BS), &H1C (->), or &H1D (<-) is output. Codes &H00 to &H09 and &H0E to &H1B are ignored (no output) at Port 1 but are output at Port 2.
	When Closed: Nothing is executed.
LPRT	Set the column position to 0 (i.e., the leftmost position) when code &H0A, &H0D, &H0B, or &H0C is output. Characters exceeding 80th character are output with code &H0A (LF) appended.
	When Closed: If characters (80 characters or less) remain in the buffer, they are output along with &H0A (LF).
COMU	If characters are input to the buffer, they are output.
	When Closed: If characters remain in the buffer, they are output.

4-2-5 Arithmetic Operation Functions

ABS Function

Purpose: To return the absolute value of the numeric expression specified by the argument

Format: ABS(<x>)

Example: A = ABS (-1.5)

ACOS Function

Purpose: To return the arc cosine of the numeric expression given by the argument

Format: ACOS(<x>)

<x> is a number in the range of -1 to 1.

Example: A = ACOS (1)

Remarks: The arc cosine is given in radian units in the range of 0 to pi.

ASIN Function

Purpose: To return the arc sine of the value given by the argument

Format: ASIN(<x>)

<x> is a number in the range of -1 to 1.

Example: A = ASIN (1)

Remarks: The arc sine is given in radian units in the range of -pi/2 to pi/2.

ATN Function

Purpose: To return the arc tangent of the value given by the argument

Format: ATN(<x>)

<x> is a number in the range of -1 to 1.

Example: A = ATN (1)

Remarks: The arc tangent is given in radian units in the range of -pi/2 to pi/2.

CDBL Function

Purpose: To convert a single-precision numeric value into double-precision

Format: CDBL(<x>)

Example: CDBL (2/3)

CINT Function

Purpose: To round off a numeric value at the decimal point and convert it into an integer

Format: CINT(<x>)

Example: A = CINT(B#)

COS Function

Purpose: To return the cosine of the numeric value given by the argument

Format: COS(<x>)

<x> is an expression in radian units.

Example: A = COS(pi/2)

CSNG Function

Purpose: To convert a numeric value into a single-precision real number

Format: CSNG(<x>)

Example: B = CSNG(C#)

FIX Function

Purpose: To return the integer part of the expression specified by the argument

Format: FIX(<x>)

Example: A = FIX(B/3)

Remarks: If the value of the argument is negative, this function returns a different value than the INF function returns.

INT Function

Purpose: To return the truncated integer of a numeric value

Format: INT(<x>)

Example: A = INT(B)

Remarks: Returns the largest integer value less than or equal to the value specified by the argument.
If the value of the argument is negative, this function returns a different value than the FIX function returns.

LOG Function

Purpose: To return the natural logarithm of the argument

Format: LOG(<x>)

<x> must be greater than 0.

Example: A = LOG(5)

RND Function

Purpose: To return a random number between 0 and 1.

Format: RND [<x>]

Example: A = RND(1)

Remarks:

If <x> is negative, a new random number is generated.

If <x> is omitted, or if it is positive, the next random number of the sequence is generated.

If <x> is 0, the last generated random number is repeated.

The sequence can be changed by executing the RANDOM statement.

SGN Function

Purpose: To return the sign of an argument

Format: SIGN(<x>)

Example: B = SGN(A)

Remarks:

If the value of <x> is positive, SGN returns 1.

If the value of <x> is negative, SGN returns -1.

If the the value of <x> is 0, SGN returns 0.

SIN Function

Purpose: To return the sine of the numeric value given by the argument

Format: SIN(<x>)

<x> is an expression in radian units.

Example: A = SIN(pi)

TAN Function

Purpose: To return the tangent of the numeric value given by the argument

Format: TAN(<x>)

<x> is an expression in radian units.

Example: A = TAN(3.141592/2)

4-2-6 Character String Functions

ASC Function

Purpose: To return the ASCII character code of the first character of the given string

Format: ASC(<x$>)

Example: A = ASC(A$)

Remarks:

An empty string cannot be specified. The word R$ function performs the inverse operation.

CHR$ Function

Purpose: To return a character corresponding to the specified character code

Format: CHR$(<i>)

Example: A$ = word R$(&H41)

Remarks:

<i> must be from 0 to 255. If <i> is a real number, it will be rounded off and converted into an integer. The ASC function performs the inverse operation.

HEX$ Function

Purpose: To return a string which represents the hexadecimal value of the decimal argument

Format: HEX$(<x>)

Example: A$ = HEX$(52)

Remarks: If the value of the decimal number includes a decimal point, the INF function is internally executed to round it off to an integer.

INSTR Function

Purpose: To return the position of the first occurrence of string <y$> within string <$x>

Format: INSTR([<i>,]<x$>,<y$>)

<i> is the position from where the search starts. <i> must be between one and 255.

<x$> is the string to be searched.

<y$> is the desired string.

Example: A = INSTR(5,B$,“BASIC”)

Remarks: If <i> is omitted, the search begins with the first character in <x$>. If the data cannot be found, 0 is returned as the function value. If <y$> is an empty string, INSTR returns <i> or 1.

LEFT$ Function

Purpose: To return the specified number of characters beginning from the leftmost character of the character string

Format: LEFT$(<x$>,<i>)

<x$> is the string to be searched.

<i> is the number of characters to be returned.

Example: A$ = LEFT$(B$,5)

Remarks: <i> must be an integer from 0 to 255. If <i> is 0, an empty string is returned as the function value. If <i> is greater than the number of characters in <x$>, the entire character string is returned.

LEN Function

Purpose: To return the number of characters in a character string

Format: LEN(<x$>)

Example: A = LEN(A$)

Remarks: A value of 0 is returned if the “character expression” is an empty string.

MID$ Function

Purpose: To return the requested part of a given string

Format: MID$(<x$>,<i>[,<j>])

<x$> is the given string.

<i> is the position of the first character to be returned.

<j> is the number of characters to be returned.

Example:B$ = MID$(A$,2,5)

Remarks:

<i> must be from 1 to 255.

<j> must be from 0 to 255.

If <j> is 0, or if the value of the specified character position (<i>) is greater than the number of characters in the character expression (x$), an empty string is returned.

If <j> is omitted, or if <j> exceeds the number of characters to the right of the specified position (<i>) in the character expression, all the characters to the right are returned.

OCT$ Function

Purpose: To convert the specified decimal number into an octal character string

Format: OCT$(<x>)

<x> is a numeric expression in the range of -32768 to 32767.

Example: A$ = OCT$(B)

Remarks:

If the value of <x> includes a decimal point, the INT function is internally executed to round it off.

RIGHT$ Function

Purpose: To return the specified number of characters from the rightmost character of the character string

Format: RIGHT$(<x$>,<i>)

<x$> is the string to be searched.

<i> is the number of characters to be returned.

Example: A$ = RIGHT$(B$,5)

Remarks:

<i> must be an integer from 0 to 255. If <i> is 0, an empty string is returned as the function value. If <i> is greater than the number of characters in <x$>, the entire character string is returned.

SPACE$ Function

Purpose: To return a string of spaces of the specified length

Format: SPACE$(<x>)

<x> is the number of spaces.

Example: A$ = “CF”+SPACE$(5)+“BASIC”

Remarks:

<x> must be from 0 to 255. If <x> is not an integer, it will be rounded off. If 0 is specified, an empty character string is returned.

STR$ Function

Purpose: Converts the specified numeric value into a character string

Format: STR$(<x>)

Example: B$ = “A”+STR$(123)

Remarks: The VAL function performs the inverse operation.

STRING$ Function

Purpose: To return a character string of the specified character and length

Formats: STRING$(<i>,<j>)

STRING$(<i>,<x$>)

<i> is the number of characters to be returned.

<j> is the ASCII code of some character.

<x$> is a given string.

Example: A$ = STRING$(10,“A”)

Remarks:

<i> and <j> must be from 0 to 255.

An empty string is returned if the <i> is 0.

If the <x$> is made up of two or more characters, only the first character is used.

TAB Function

Purpose: To move the cursor to a specific position on the terminal display

Format: TAB(<i>)

<i> is the cursor position counting from the leftmost side of the display.

Example: PRINT “CF” TAB (10) “BASIC”

Remarks:

The “column position” must be from 1 to 255.

If the current print position is already beyond <i>, the cursor moves to the <i>th position on the next line. TAB is only valid for the PRINT and LPRINT statements.

VAL Function

Purpose: To convert a character string into a numeric value

Format: VAL(<x$>)

Example: A = VAL(A$)

Remarks:

The VAL function also strips leading blanks, tabs, and linefeeds from the argument string. If the first character of <x$> is not numeric, zero is returned.

4-2-7 Special Functions

DATE$ Function

Purpose: To set or display the current date

Format: As a statement: DATE$ = <x$>

As a variable: <y$> = DATE$

<x$>: the date in one of the following formats:

mm-dd-yy
mm-dd-yyyy
mm/dd/yy
mm/dd/yyyy

<y$>: A ten character string in mm-dd-yyyy format:

mm: two digit value for the month (01-12)
dd: two digit value for the day (01-31)
yy: two digit value for the year
yyyy: for digit value for the year

Example: DATE$ = “89/05/23”

Remarks:

If DATE$ is on the right side of the assignment statement or in a PRINT statement, the current date is assigned or printed, respectively. If DATE$ is on the left side of the assignment, the right side of the assignment statement becomes the new current date. If any of the values are out of range or are missing, an error message will be displayed.

DAY Function

Purpose: To give or set the current day of the week

Format: DAY = <num>
I = DAY

Remarks:

In the first format, DAY returns a number between 0 and 6, corresponding to Sunday through Saturday. In the second format, the day of the week is assigned to DAY.

EOF Function

Purpose: To check whether the specified port buffer is empty

Format: EOF (<port#>)

Example: IF EOF (2) THEN CLOSE#1 ELSE GOTO 100

Remarks:

This function returns true (-1) if the specified port is empty. If not, it returns false (0). Note that the port specified by <port#> must already be open and in the input mode.

ERR and ERL Variables

Purpose: To return the error code and the location (line number) of the error

Format: x = ERL

y = ERR

Remarks:

When an error occurs, the error code is assigned to the variable ERR and the statement number is assigned to ERL.

If the statement that caused the error was executed in direct mode, statement number 65535 is assigned to ERL.

ERL and ERR can be used in error handling routines to control the execution flow of the program.

FRE Function

Purpose: To return the amount of unused memory

Format: FRE(0)

FRE(<x$>)

Example: PRINT FRE (0)

Remarks:

If the argument is numeric, the number of unused bytes in the program area is given.

If the argument is a character expression, the number of unused bytes in the character variable area is given.

When this instruction is executed, the unnecessary parameter area will be filled.

To avoid long interruption times, execute this instruction intermittently so that each interruption will be a short one.

INKEY$ Function

Purpose: To return the character code of the key being pressed

Format: INKEY$ [#<port>]

Example: A$ = INKEY$

Remarks:

A null string is returned if no key is being pressed. Any key input other than CTRL+X is valid. Port 1 is the default port.

INPUT$ Function

Purpose: To Read a string of characters from the keyboard or from a peripheral device

Format: INPUT$ (<num>[,#<port>])

<num> is the number of characters to be input. <num> must be from 1 to 255.

<port> is the port number (1 or 2).

Example: A$ = INPUT$(10,#1)

Remarks:

All characters except CTRL+X can be read, including CR and LF: CR and LF cannot be read with the LINE INPUT statement.

The BASIC LED indicator on the ASCII Unit will blink indicating that the unit is waiting for input. It will continue blinking until the specified number of characters is entered.

Example Program:

```
10  CLS
20  A$ = INPUT$ (1)
30  A$ = HEX$ (ASC(A$))
40  PRINT A$
50  GOTO 20
```

Remarks:

displays key character codes.

LOC Function

Purpose: To return the number of data items in the specified port buffer.

Format: x = LOC(<port#>)

Example: A = LOC(2)

Remarks:

The port specified must already be open and in input mode. The number of data items in the buffer of the specified port is given in byte units.

PEEK Function

Purpose: To read the contents of a specified memory address

Format: PEEK(<I>)

<I> is the memory location and must be in the range of 0 to 65535 (&HFFFF).

Example: A = PEEK(&H3000)

Remarks:

If the specified address is not an integer, it is converted into one.

Do not try to read reserved system addresses &H0000 to &H1FFF and &H8000 to HFFFF.

TIME$ Function

Purpose: Sets or gives the time

Format: TIME$ = <x$>

<y$> = TIME$

<x$> is a string expression indicating the time to be set. The following formats may be used:

hh: sets the hour (minutes and seconds 00)

hh:mm: sets the hours and minutes (seconds 00)

hh:mm:ss: sets the hours, minutes, and seconds

<y$> is a string variable to which the current value of the time is to be assigned.

Example: TIME$ = “09:10:00”

PRINT TIME$

Remarks:

In the form <y$> = TIME$, TIME$ returns an eight character string in the form: hh:mm:ss. If <x$> is not a valid string, an error message will be displayed.

USR Function

Purpose: To call a user-written assembly language program.

Format: USR [<number>](<argument>)[,W]

<number> is a digit from 1 to 9 that was previously assigned to the given assembly program with the DEF USR statement.

<x> is an argument used to pass data from the BASIC program to the assembly program.

Example: J = USR2(I),W

Remarks:

If <number> is omitted, the default value is zero.

If the W parameter in the USR statement is not specified then the watchdog timer refresh will be performed as usual. If the W parameter is specified, then the user must include a watchdog timer refresh routine in the assembly program.

The watchdog timer prevents the program from overrunning. When the set time has run out, the ASCII Unit is reset, and the message “I/O ERR” is displayed on the programming console of the PC.

By refreshing the watchdog timer before its set value is up, the program can be continuously executed.

To refresh the watchdog timer in the assembly program, execute the following two steps every 90 milliseconds when, W has been designated:

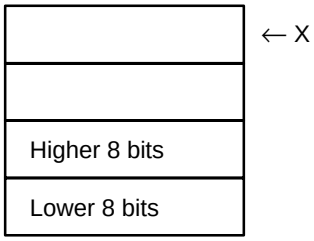
AIM #DF,03
OIM #20,03

The following table lists the Argument type and its corresponding Accumulator code number.

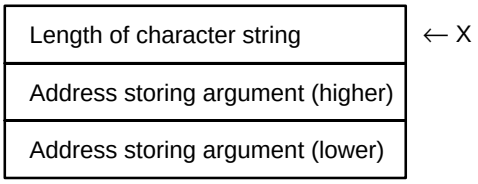
Accumulator Value	Argument Type
2	Integer
3	Character
4	Single-precision, real number
5	Double-precision, real number

Index register X contains the memory address where the argument is stored. The address differs as shown below depending on the type of the argument.

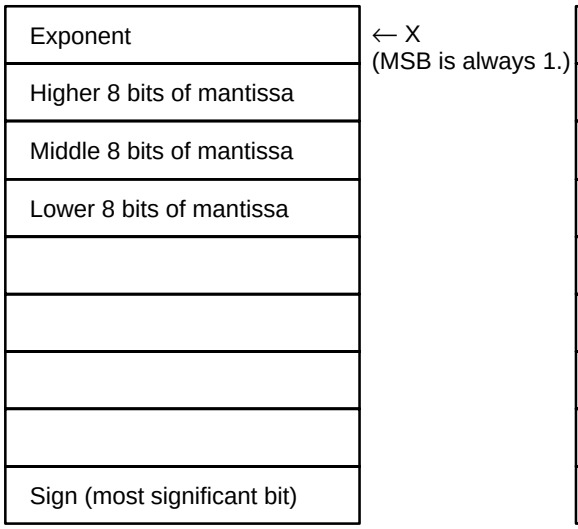
Integer Type



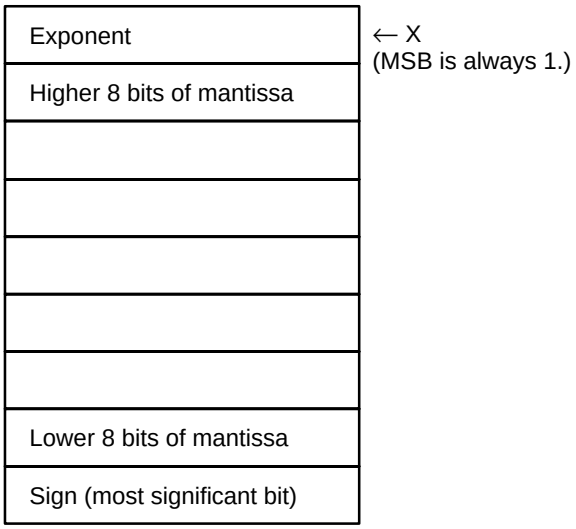
Character Type



Single-Precision, Real Number Type



Double-Precision, Real Number Type



Program Example:

BASIC Program:

```
100     A$ = &H1234
110     DEF USR0 = &H2000
120     A = USER (A)
130     PRINT A
140     END
```

Assembly Language Program:

```
2000    PSHA
2001    PSHX
2002    LDD 2,X
2004    ADD #10
2007    STD 2,X
2009    PULX
2010    PULA
2011    RTS
```

Program Remarks:

When program execution branches to the assembly language routine, the TYPE of <argument> is stored in the accumulator A, and the memory address where the argument is stored is input to the index register X. The value of the argument is stored in the accumulator D, to whose contents 10 will be added. The result of the addition is written to the address of <argument>

VARPTR Function

Purpose: Returns the memory address of the variable argument

Format: <x> = VARPTR(<variable>)

<variable> is a number, string, or array variable.

Example: B = VARPTR (A)

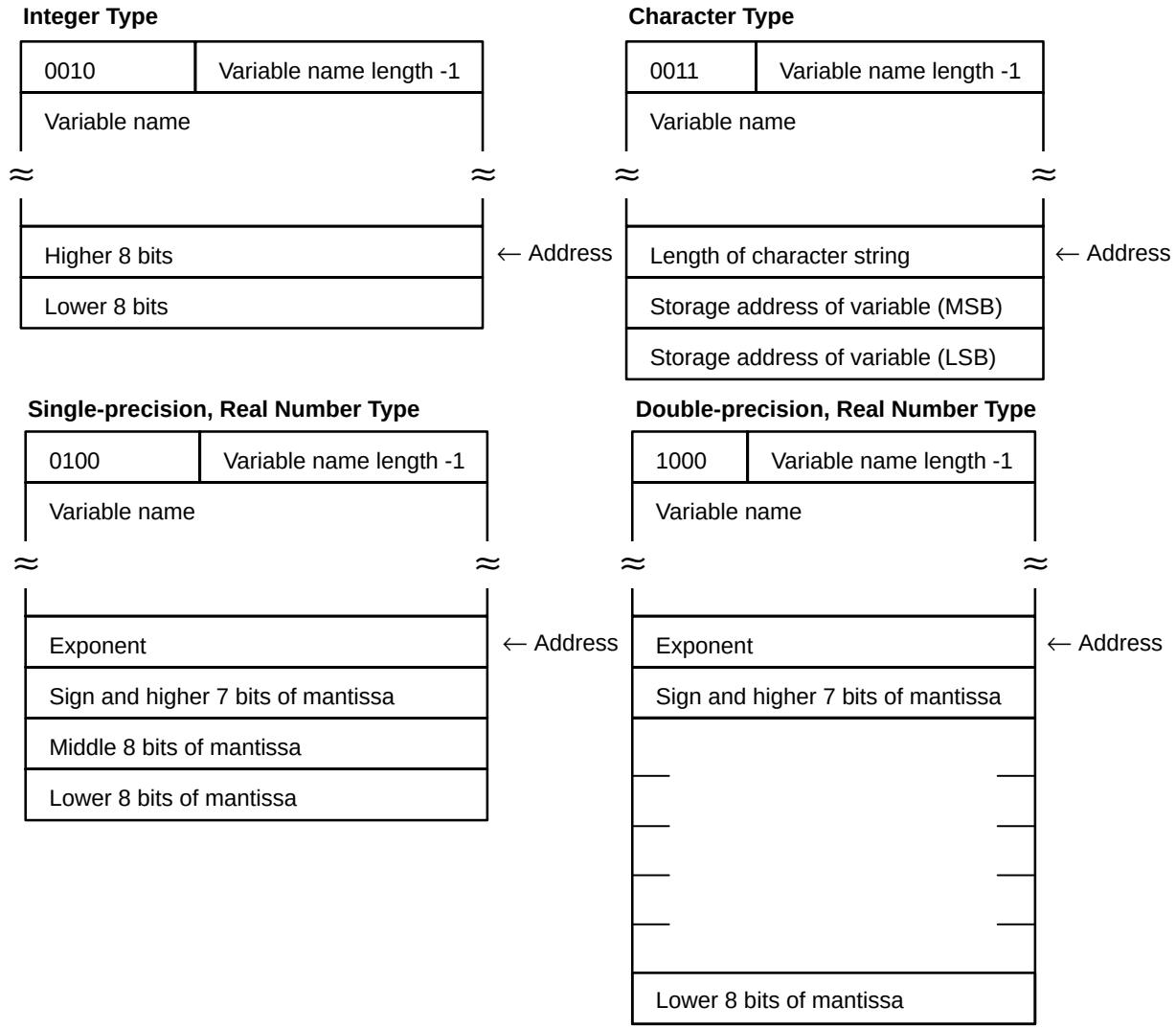
Remarks:

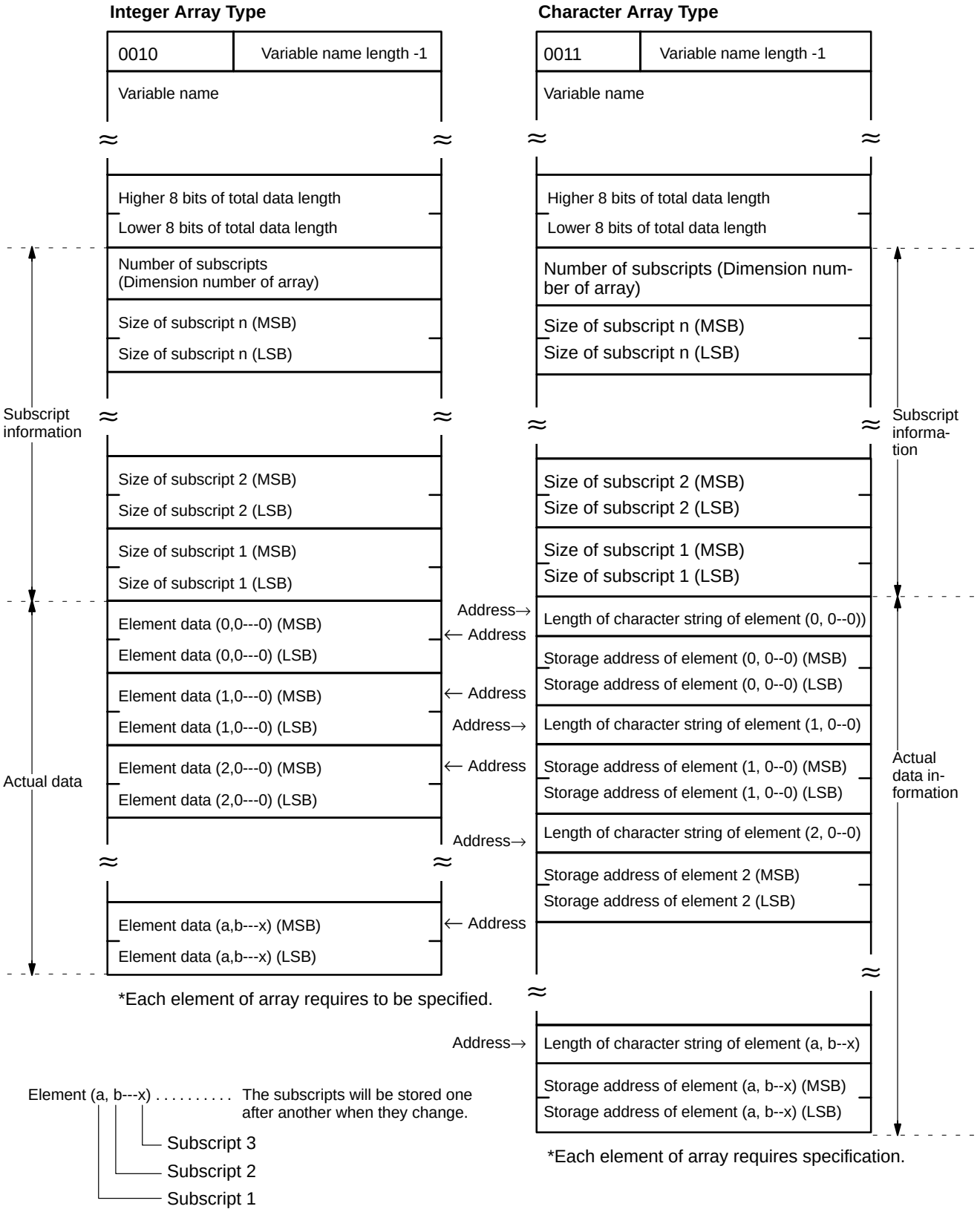
The VARPTR function returns the address of the first byte of data identified with the variable. A value must be assigned to the variable prior to the call to VARPTR or an error will result. Any type variable name may be used (numeric, string, array).

Note that all simple variables should be assigned before calling VARPTR for an array because addresses of arrays change whenever a new simple variable is assigned.

VARPTR is used to obtain the address of a variable or array so that it may be passed to an assembly language subroutine. A function call of the form VARPTR(A(0)) is specified when passing an array, so that the lowest addressed element of the array is returned.

The following figure illustrates the relationship between the variable type and the address indicated by VARPTR.





Note The total number of bytes from the higher 8 bits of total data length (in the above diagram) to element data (a,b---x) comprise the total length of the data.

SECTION 5
Assembly Programming

This section explains how to create, edit, transfer, and use an assembly language program. Assembly programs are faster and use memory more efficiently than higher level programs such as BASIC. In certain situations it is advantageous to use assembly routines instead of BASIC to perform specialized functions. An assembly routine can be called from the BASIC program and used in much the same way as a BASIC subroutine.

Assembly programs are written, edited, and tested in what is called *Monitor Mode*. The monitor mode commands and examples of their use are presented in this section.

5-1 Assembly Language Programming

The Hitachi HD6303X CPU is incorporated into the ASCII Unit. Mnemonics used are those found in the HD6303X operation manual.

Memory Area

Special memory space for assembly language programs must be reserved with the MSET command. When programming in assembly language, you cannot use the BASIC program area to store the assembly program. The MSET command will move an existing BASIC program to another part of memory.

Writing an Assembly Program

There are two ways to write an assembly language program:

- By using the monitor functions
- By directly writing the program to the memory using the POKE statement in BASIC.

In most cases the first method is quicker and easier, however, the second method can be used to create short programs consisting of only a few steps.

Assembly language programs can be written to and read from RAM using the S and L commands, respectively. They can also be written to or read from the EEPROM by using the SAVE and LOAD commands, respectively.

Addresses &H0000 to &H1FFF and &H8000 to &HFFFF are reserved for the ASCII Unit operating system and must not be altered by the user.

Note When it is necessary to load or save data using a peripheral device other than the input terminal connected to port 1, follow the *peripheral data transfer procedure* described below.

1, 2, 3...

1. Enter the command and key in a carriage return.
2. Disconnect the input terminal from port 1 and connect the peripheral device.
3. Press the START/STOP switch on the ASCII Unit to start data transfer.
4. Reconnect the input terminal and key in CTRL+x.

The Assembly Language Program

An assembly language program can be called from BASIC with the USR function:

USR [<number>][<argument>]

Before the USR function can be used, the DEF USR statement must be executed to reserve space for the assembly routine. When the USR function is executed, it calls the specified assembly routine and passes it an argument defined in the BASIC program.

Variables other than the argument specified by the USR function can also be passed to the assembly language program by using the VARPTR function.

The following arguments are passed to the assembly program:

Accumulator A contents: type of <argument>

Index register X contents: address of <argument>

The RTS command should be the last command of the assembly routine; it returns execution back to the BASIC program.

The value of the stack pointer must not be altered by the assembly routine. Therefore, the data should be pushed on the stack at the beginning of the routine and then pulled off before executing the RTS command.

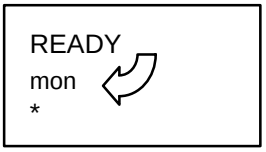
The assembly routine must store any data needed by the BASIC program in the same address as that of the argument(s) passed by the USR or VARPTR functions. Any data passed back to the BASIC program must be of the same TYPE as the USR or VARPTR Function argument(s).

Do not disable any interrupts in the assembly language program.

It is recommended that the assembly language program be saved on an external storage device or in the EEPROM for safety.

Monitor Mode

To enter monitor mode from BASIC mode, key in “mon” followed by a carriage return when the message “READY” is displayed on the console:



When in monitor mode a “*” is displayed on the left-side of the screen. Also, when in monitor mode, the BASIC LED on the ASCII Unit front panel is unlit.

To return to BASIC mode, key in CTRL+B.

5-2 Terminology and Formatting

Terminology

Start address refers to the first memory address where a group of values stored in consecutive memory locations is stored: e.g., an array or a block of data.

For some monitor mode commands, indicating a start address is optional. For these commands, the address immediately following the highest or largest address used by the previous monitor command is taken to be the start address for the current monitor command. To simplify following explanations, this address will be called the **base** address.

An assembly language program can be edited, traced, and debugged in **monitor mode**.

Note that the address held in the **program counter** is the base address used for displaying and writing data when using the monitor commands.

Format

The left and right arrow brackets “<” and “>” that have been previously used to denote “user supplied text” in BASIC programming format statements are used as actual operators in monitor mode. Therefore, whenever you see an arrow bracket character in a monitor mode command, it **must be entered as such.** The arrow character is used to delineate address ranges.

For monitor format statements only, left and right parentheses “( )” will be used to denote user supplied text.

Brackets “[ ]” still indicate optional entry. Pay close attention to periods “.”; **they must be entered as such** whenever indicated.

The carriage return key is indicated with ↵. Whenever this appears in a command, a carriage return must be entered by the user.

Do not insert spaces within a monitor command unless explicitly indicated.

In the following examples and also on the actual terminal the “*” character indicates that the user must enter a command. Lines of text that do not start with a “*” are generated by the computer in response to a user command.

5-3 Monitor Mode Commands

The following table lists the monitor mode commands with a short description of each command’s function as well as the page number on which its detailed explanation can be found.

To enter monitor mode, type mon and carriage return at the READY prompt.

Note Enter all command in all-caps while in monitor mode. Do not use lower case.

Page	Command	Purpose
70	address	Displays/changes memory contents at the specified address.
71	M	Transfers memory contents.
72	C	Compares memory contents.
72	R	Displays/changes register contents.
73	BP	Sets/displays break points.
73	N	Clears break points.
73	I	Disassembler
74	S	Outputs data to a port.
74	L	Loads data from a port.
75	V	Verifies data.
75	G	Executes a program.
75	T	Single-step program execution
76	Mini-assembler	Single-line assembly
76	Arithmetic	Addition/subtraction of hexadecimal numbers.

DUMP Command

Purpose: To display the contents of memory in hexadecimal

Format: [(display start address)].[(display end address)]

Remarks:

If the carriage return ↵ is input by itself, eight bytes of data, starting from the base address will be displayed (refer to example 2.)

If an address is entered preceded by a dot, e.g., ".3000", data stored in all the addresses from the base address to the entered address will be displayed (refer to examples 3 and 4.)

New data can be stored in memory as well; this data will overwrite existing data. Input data must be in hexadecimal. Upper case characters must be used for the alphanumeric values of A to F (hex). When the leftmost digit is a "0", it can be omitted.

There are two ways to poke data (directly store data to a specific address).

1, 2, 3...

1. Specify the first address followed by a colon. Directly after the colon, enter the data (1 or 2 byte hexadecimal values only) separated by spaces. Then type a carriage return (refer to example 5.)
2. Enter a colon followed by the data and type a carriage return. Data will be stored starting from the base address (refer to example 6.)

Examples:

1. Enter: *4000 ↵
   Displayed: 4000–10
   - Displays 1 byte of data from the specified address.
2. Enter: * ↵
   Displayed: *20 30 50 60 70 80 90 9F
   - Displays 8 bytes of data, starting from the base address.
3. Enter: *.4010A ↵
   Displayed: 4008–A0 B0 C0 D0 E0 F0 00 10
   4010–01 02 03 04 05 06 07 08
   4018–12 34 56
   - Displays all of the data from the base address to the specified address.

4. Enter: *.3000 ↵
   Displayed: 401B–78
   - If the “dot” address format is used and the entered address is lower than the base address, the contents of the specified address will not be displayed. The contents of the base address will be displayed instead.
5. Enter: *3000:9 8 7 6 5 4 3 2 1 ↵
   *3000.3007 ↵
   Displayed: 3000–09 08 07 06 05 04 03 21
   - Pokes data in a series of addresses starting from the specified address.
6. Enter: *:11 22 33 44 55 ↵
   *3000.3007 ↵
   Displayed: 3000–11 22 33 44 55 04 03 21
   - Pokes data in a series of addresses starting from the base address.

Move Command

Purpose: To transfer the data stored in a consecutive range of addresses to another place in memory

Format: M(destination start address)< (source start address). (source end address)

Remarks:

This command will transfer a block of data starting from (source start address) and ending at (source end address) to (destination start address). Note that the source address range must not overlap the destination address range; otherwise, the data will not be transferred correctly.

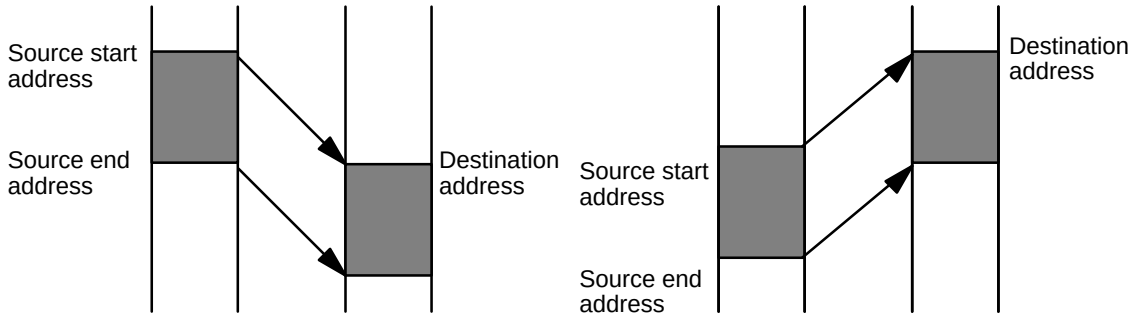
Example:

Enter: *M3000<4000.4007 ↵
*4000.4007 ↵
Displayed: 4000–01 02 03 04 05 06 07 08
Enter: *3000.3007 ↵
Displayed: 3000–01 02 03 04 05 06 07 08

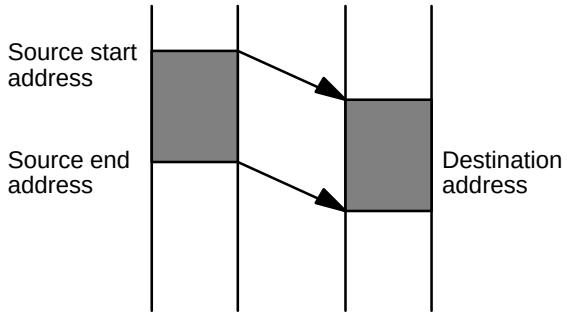
Example Remarks:

In the above example, the contents of addresses 4000 to 4007 are transferred to an address range starting at address 3000.

The following diagram illustrates correct and incorrect usage of the Move command.



Proper Data Movement



In this example, the source start address is smaller than the destination address and the destination address is equal to or smaller than the source end address. Consequently, the data is not transferred correctly. Transfer the data to an area that has not been overlapped and transfer the data again.

Improper Data Movement

Compare Command

Purpose: To compare two blocks of data

Format: (start address 1)<(start address 2).(end address 2)

Remarks:

Compares the data stored from (start address 2) to (end address 2) to a block of data of the same size starting at (start address 1). If the contents of the two address ranges differ, the corresponding address(es) where the data is not the same is displayed with its contents.

Example:

Enter:	*C3000<4000.4007 ↵
Displayed:	4003–FF (03)
Enter:	*3000.3007 ↵
Displayed:	3000–00 01 02 03 04 05 06 07
Enter:	*4000.4007 ↵
Displayed:	4000–00 01 02 FF 04 05 06 07

Example Remarks:

In the above example, data stored in addresses 3000 to 3007 is compared with data stored in addresses 4000 to 4007. In this example, the data stored in address 3003 has been found to differ from the data stored in address 4003. Consequently, the data stored in address 4003 (FF) and the data stored in address 3003 (03) are displayed.

Register Command

Purpose: To display or change the contents of a register.

Format: R(register) = (data)
(register) is one of the hardware registers: C, A, B, X, S, or P.
(data) is a one or two digit hexadecimal number.

Remarks:

If R is entered by itself, all of the registers and their contents will be displayed.

Examples:

1. Enter: *R ↵
   Displayed: C–C0 A–00 B–01 X–ABCD S–2EFF P–5000
   - The contents of all the registers are displayed.
2. Enter: *A=12 ↵
   *X=FF00 ↵
   *R ↵
   Displayed: C–C0 A–12 B–01 X–FF00 S–2EFF P–5000
   - The contents of the specified registers (A and X) are rewritten as specified.

Break Point Command

Purpose: To set a break point at a specified address

Format: BP[(address)]

Remarks:

Up to two break points can be set at the same time. If BP is entered by itself, the current break point(s) will be displayed. If BP is followed by an address, a new break point will be set at that address.

Examples:

1. Enter: *BP3000 ↵
   - Sets a Break point.
2. Enter: *BP ↵
   Displayed: BP=3000
   - Displays the currently set bread points.
3. Enter: *BP5000 ↵
   *BP ↵
   Displayed: BP=5000 3000
   - Up to two bread points can be set.

New Command

Purpose: To clear all bread points.

Format: N

Example:

Enter: *N ↵

*BP ↵

Displayed: BP=0000 0000

Example Remarks:

Clears all the bread points currently set.

Disassembler Command

Purpose: To disassemble and display 20 lines of code starting from the specified address.

Format: I(address)

Examples:

1. Enter: *I 3000

   Displayed:
   ```
   3000–CE 10 00 LDX   #$1000
   3003–FF 40 00 STX   $4000
   3006–86 80    LDAA  #$80
    .            .     .
    .            .     .
   3030–81 12    CMPA  #$12
   ```

   - Disassembles and displays 20 lines of code starting from the specified address.

2. Enter: *I, I ↵

   Displayed:
   ```
   3032–26 02    BNE   $3036
   3034–A7 00    STAA  $00, X
   3036–39       RTS
    .            .
    .            .
   3080–08       INX
   ```

   - Each time I,I is subsequently entered, the next 20 lines of code will be displayed.

Save Command

Purpose: To transfer the specified block of data to port 1 in S format

Format: S(start address).(end address)

Remarks:

Transfers the data stored from (start address) to (end address) in S format to the port 1 buffer.

Example:

Step 1: *S3000.300F ↵

Step 2: Press the START/STOP switch.

Example Remarks:

The data stored from &H3000 to &H300F will be transferred to port 1. If a peripheral device other than the input terminal needs to be connected for the data transfer, follow the peripheral data transfer procedure explained at the beginning of this section.

Load Command

Purpose: To load a data file in S format through port 1

Format: L[(offset)]

Examples:

1. Enter: *L ↵

   Enter: *L100 ↵

   Press the START/STOP switch.

   - Loads a data file in S format through port 1 and stores the file in memory.

2. Enter: *3100.310F ↵

   Displayed:
   ```
   3100–CE 00 00 08 26 FD 08 26
   3108–FD 86 55 97 17 CE 00 00
   ```

   - When an offset address is specified, the loaded file is stored in memory starting from an address whose value is the **specified address** plus the

offset. Data transfer will not start until the ASCII Unit START/STOP switch is pressed.

Verify Command

Purpose: To verify whether data sent through port 1 is the same as data stored in the specified memory locations

Format: V[(offset)]

Example:

Enter: *V100 ↵
Press the START/STOP switch.

Displayed: 3120–12

Remarks:

The input data is compared with the data stored in the specified address range. The base address for data comparison is the **specified address** plus the offset.

If a discrepancy is found, the address at which it occurs and the data contained therein are both displayed. Data will not be verified until the ASCII Unit START/STOP switch is pressed.

If a peripheral device other than the input terminal needs to be connected for data transfer, follow the peripheral data transfer procedure explained at the beginning of this section.

Go Command

Purpose: To execute a program

Format: G[(address)]

Example:

Enter: *I3000 ↵

Displayed:
```
3000–86 80     LDAA  #$80
3002–B7 40 00  STAA  $4000
3005–20 F9     BRA   $3000
```

Enter: *BP3005 ↵
*G3000 ↵

Displayed: C–C8 A–80 B–FF X–0000 S–2EFF P–3005

Remarks:

If an address is specified, the user program is executed starting from that address. If no address is specified, execution will start from the address indicated by the program counter.

If program execution is aborted due to a bread point, SW1, or an interrupt, the register contents will be displayed.

If the stack pointer is not set to the assembly language area, this command will not execute correctly.

Step Command

Purpose: To execute a program one step at a time. This command is used for debugging.

Format: T[(address)]

Example:

Enter: *T3000 ↵

Displayed:
```
3000–86 80     LDAA  #$80
C–C8 A–80 B–00 X–0000 S–2EFF P–3002
```

Remarks:

When (address) is specified, the instruction stored starting at (address) is executed. If (address) is not specified, the instruction stored at the address indicated by the program counter is executed. To execute several program steps, execute the Step command as many times as required.

When Step is executed, the instruction stored at the specified address is displayed as well as the contents of all the hardware registers.

Mini-assembler

Purpose: To assemble one line of the program at a time.

Note Mnemonics used in Hitachi's HD6303X CPU operation manual are used here.

Procedure:

1, 2, 3... 1. Key in CTRL+A
2. Type in one line of code and a carriage return.
3. To stop, key in X followed by a carriage return.

Remarks:

Keying in CTRL+A puts the monitor in mini-assembler mode. Each time a line of code followed by a carriage return is subsequently entered, the mini-assembler will assemble and display it. To exit mini-assembler mode enter "x" followed by a carriage return.

Note Always enter a space after the prompt (!) when using command without addresses. Always enter a space between operands.

Example:

```
Enter:      *CTRL+A ↵
            !_3000:LDAA_ #$80 ↵
Displayed:  3000–86 80    LDAA   #$80
Enter:      !_LDAB_#$7F ↵
Displayed:  3002–C6 7F    LDAB   #$7F
Enter:      !_STD_$4000 ↵
Displayed:  3004–FD 40 00 STD    $4000
Enter:      !_ASLA ↵
Displayed:  3007 48 ASLA
Enter:      !_BNE_$3000 ↵
Displayed:  3008 26 F6    BNE    $3000
Enter:      !X ↵
```

Arithmetic Using Hexadecimal

Purpose: To add or subtract 4-digit hexadecimal data.

Format: (hex data)+(hex data)
(hex data)–(hex data)

Examples:

```
Enter:      *1234+5678 ↵
Displayed:  1234+5678=68AC
Enter:      *ABCD+EF01 ↵
Displayed:  ABCD+EF01=9ACE
Enter:      *AB–12 ↵
Displayed:  AB–12=0099
```

SECTION 6
Program Examples

In order for the PC and the ASCII Unit to communicate with each other, both an ASCII Unit program written in BASIC and a PC program must be prepared. These two programs work with each other to coordinate the timing of communications and data transfer between the two devices.

The ASCII Unit can be set in one of two modes: two-word mode or four-word mode. If the ASCII Unit is set in two-word mode, the PC can use READ(88/190) and WRIT(87/191) for data transfer with the ASCII Unit. If the ASCII Unit is set in four-word mode, the PC must use the MOV(21/030) instruction to transfer data with the ASCII Unit.

The first part of this section presents an explanation of the timing between the ASCII Unit and the PC when READ(88/190) and WRIT(87/191) are used with the PC READ, PC WRITE, PC GET, and PC PUT statements. In order to understand the programming examples in this section, it is necessary to fully understand the timing explained in this section. Please study this section carefully before going on to the examples.

The second part of this section presents example programs written for the ASCII Unit and PC with the ASCII Unit set in two-word mode.

The third part of this section presents example programs written for the ASCII Unit and PC with the ASCII Unit set in four-word mode.

The fourth and last part of this section presents an assembly language programming example.

Some of the examples also present detailed explanations of what the PC and ASCII Unit are doing during execution of each devices respective programs. When this material is present, it is listed under the heading *Execution Sequence*.

6-1 Timing Considerations

READ(88/190) is the I/O READ instruction and WRIT(87/191) is I/O WRITE instruction. These are PC commands and are executed from within the PC ladder diagram program. READ(88/190) and WRIT(87/191) give the PC the ability to transfer large blocks of data during one cycle time: up to 255 words at a time. The MOV(21/030) instruction can only transfer one word of data per cycle.

Because variable sized blocks of data can be transferred with one READ(88/190)/WRIT(87/191) instruction, the amount of time needed to complete execution of the READ(88/190)/WRIT(87/191) instruction will vary depending on how many words of data are being transferred. Therefore, the PC must have a method of informing the ASCII Unit when the data transfer operation is completed. The PC uses the Equals Flag for this purpose. When the PC is in the midst of executing a READ(88/190)/WRIT(87/191) instruction, this flag is turned OFF. When the READ(88/190)/WRIT(87/191) instruction finishes executing, this flag is turned ON.

The diagram on the following page illustrates the timing relationships between READ(88/190) and the PC WRITE statement and WRIT(87/191) and the PC READ statement.

Whenever the ASCII Unit is writing data with the PC WRITE statement, the PC is reading data with READ(88/190) and whenever the PC is writing data with WRIT(87/191), the ASCII Unit is reading data with the PC READ statement. This illustrates two important points:

- Whenever the ASCII Unit and the PC communicate, one of them is reading and the other one is writing.
- The device which is writing data always initiates data transfer.

The following rules are illustrated in the diagram on the next page:

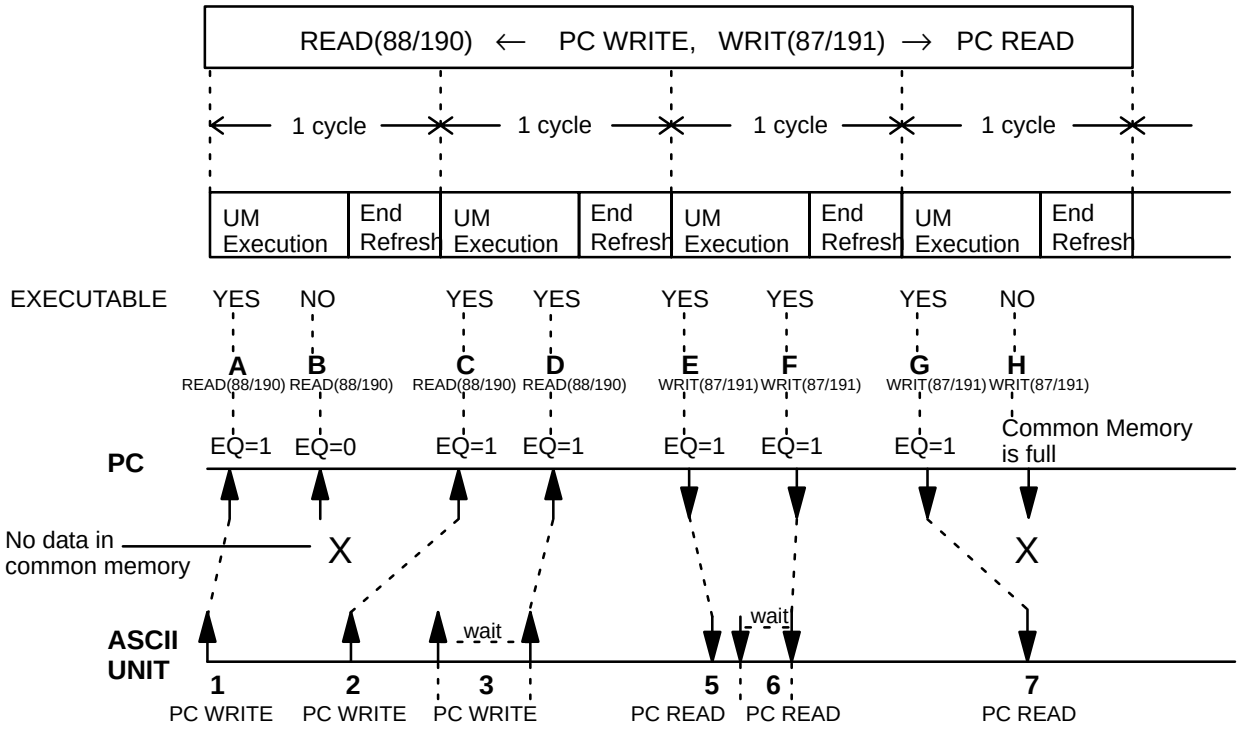
B. If a READ(88/190) is executed before its corresponding PC WRITE statement, it is treated as a NOP.

3. If a PC WRITE statement is executed before processing of a previous PC WRITE statement is completed, it must wait for execution of the next READ(88/190) before data transfer can begin.

6. If a PC READ statement is executed before processing of a previous PC READ statement is completed, it must wait for the next WRIT(87/191).

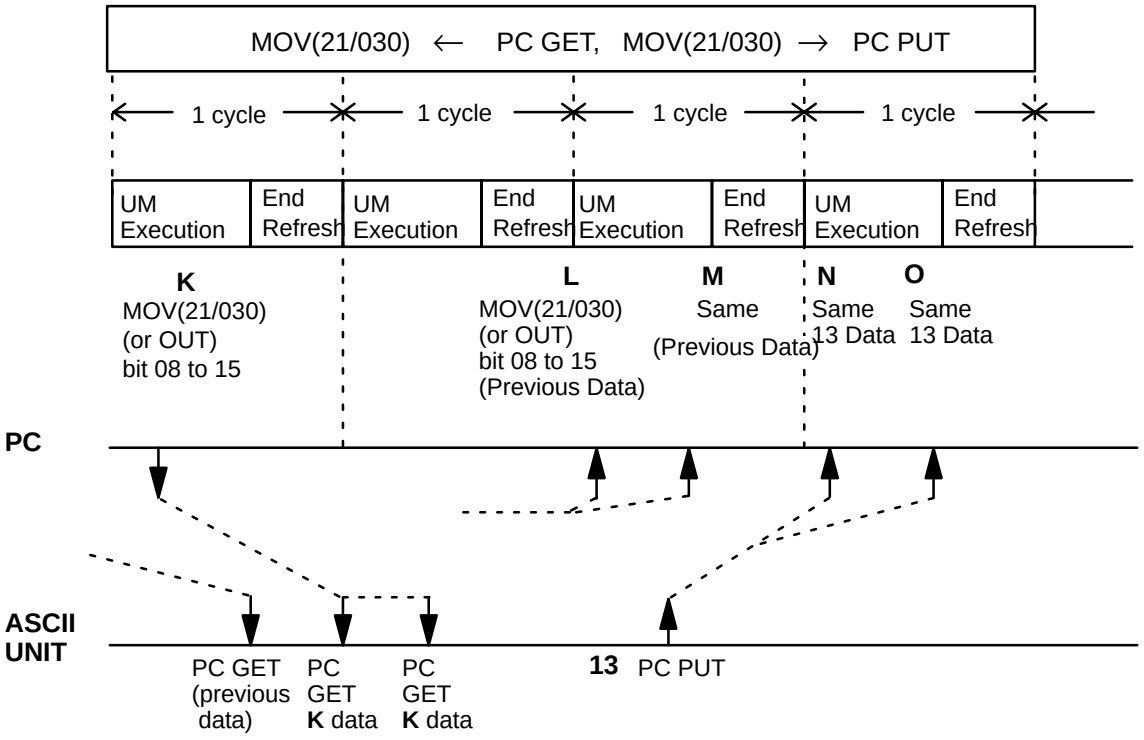
H. If a WRIT(87/191) instruction is executed before processing of the previous WRIT(87/191) instruction is completed, it is treated as a NOP.

Timing Between PC and ASCII Unit Instructions



3. Waits until data previously written to the common memory is written to the PC.

6. Waits until the data being read is transferred to the common memory



6-2 Programs in Two-word Mode

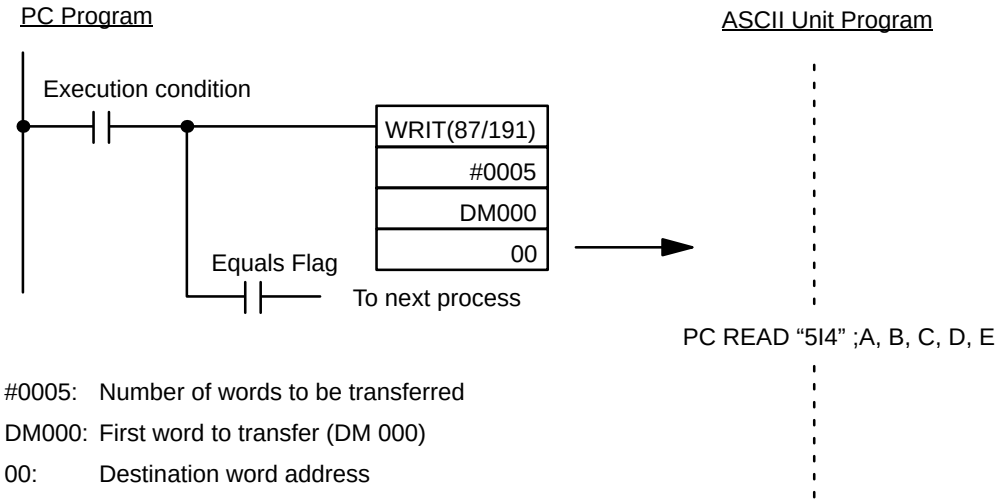
The following programs are executed with the ASCII Unit set in two-word mode.

For all of the following examples:

- printer is connected to port 2
- 8 bits/ no parity/ 2 stop bits

Example 1

Purpose: To write data from the PC using WRIT(87/191) and to the ASCII Unit using the PC READ statement.



#0005: Number of words to be transferred

DM000: First word to transfer (DM 000)

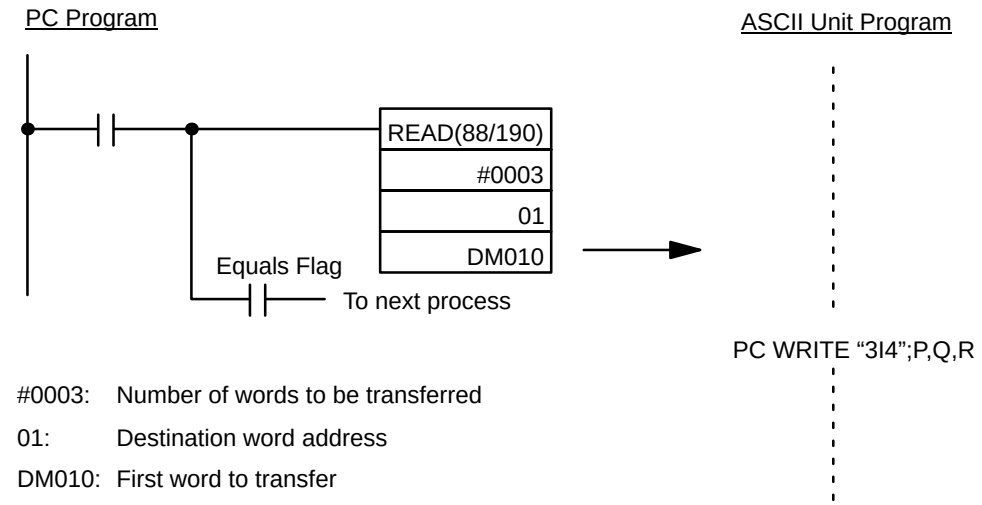
00: Destination word address

Remarks:

When the execution condition goes ON, WRIT(87/191) is executed. The ASCII Unit reads five words of data starting at DM 000, converts them into BCD, and assigns them to the variables A through E. When execution of WRIT(87/191) is completed, the Equals Flag is turned ON.

Example 2

Purpose: To write data from the ASCII Unit using the PC WRITE statement to the PC using the READ(88/190) instruction.



#0003: Number of words to be transferred

01: Destination word address

DM010: First word to transfer

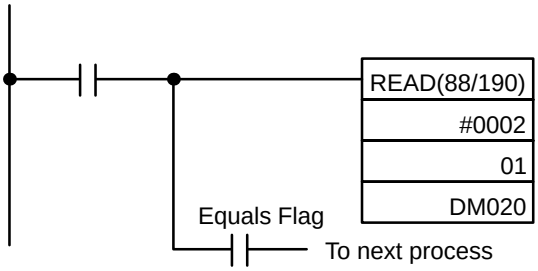
Remarks:

When the ASCII Unit executes the PC WRITE statement, the variables P, Q, and R are converted into BCD and stored in DM 010, 011, and 012.

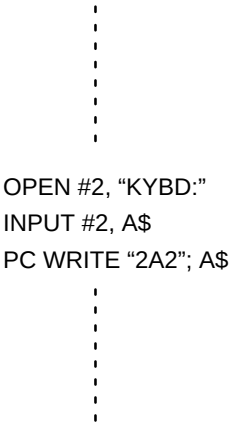
Example 3

Purpose: To enter characters from the keyboard and write them to the PC using the PC WRITE statement and READ(88/190).

PC Program



ASCII Unit Program



#0002: Number of words to be transferred

01: Destination word address

DM020: First word to transfer

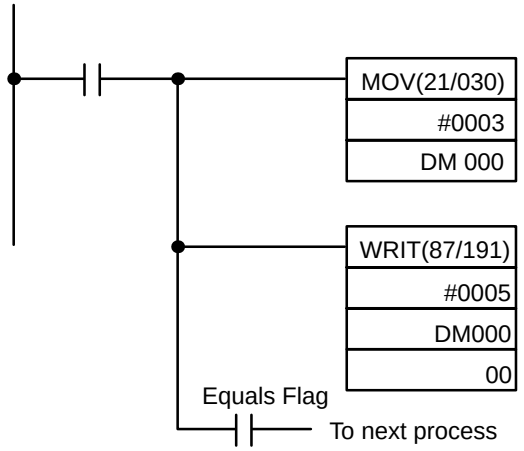
Remarks:

When the PC WRITE statement is executed, the first four characters of character string A$ are converted into ASCII code and stored in DM 0020 and 0021.

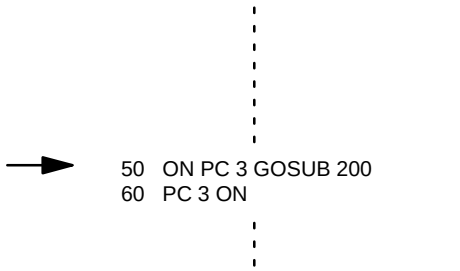
Example 4

Purpose: The PC uses interrupt number 3 to direct the ASCII Unit to read five words of data from the specified DM addresses.

PC Program



ASCII Unit Program



#0005: Number of words to be transferred

DM000: First word to transfer (DM 000)

00: Destination word address

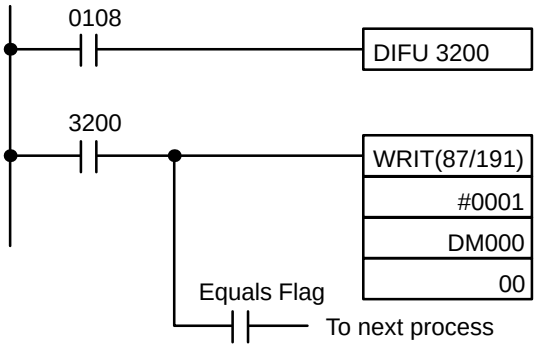
Remarks:

When the Interrupt Input goes ON, the PC writes the interrupt number to DM 000 with the MOV(21/030) instruction and the ASCII Unit branches to the interrupt service routine at line 200. WRIT(87/191) then writes 5 words of data to the ASCII Unit which stores them in variables “A” through “E”.

Example 5

Purpose: To read and print PC data at specific times using the ASCII Unit PC READ statement and WRIT(87/191)

PC Program



ASCII Unit Program

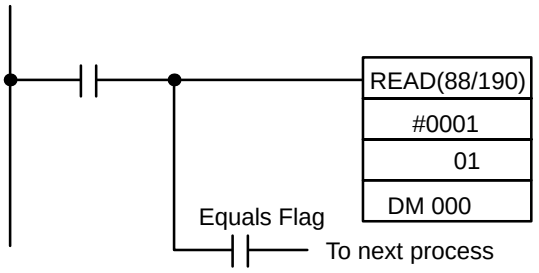
```
10  OPEN #2,"LPRT:(47)"
20  A$ =  "00:00"
30  B$ = MID$ (TIME$, 4, 5)
40  IF B$ <:> A$ GOTO 30
50  PC PUT 1
60  PC READ, "I4" ; X
70  PC PUT 0
80  PRINT #2, "DM=" ; X
90  GOTO 30
```

#0001: Number of words to be transferred
DM000: First word to transfer (DM 000)
00: Destination word address

Example 6

Purpose: To accept input from the keyboard and write it to the PC using the PC WRITE statement and READ(88/190)

PC Program



ASCII Unit Program

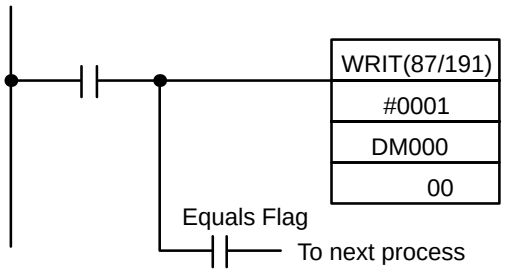
```
10  INPUT I
20  PC WRITE "I4"; I
30  GOTO 10
```

#0001: Number of words to be transferred
01: Destination word address
DM000: First word to transfer

Example 7

Purpose: To display the state of PC bit 1000 on a display device connected to port 2

PC Program



ASCII Unit Program

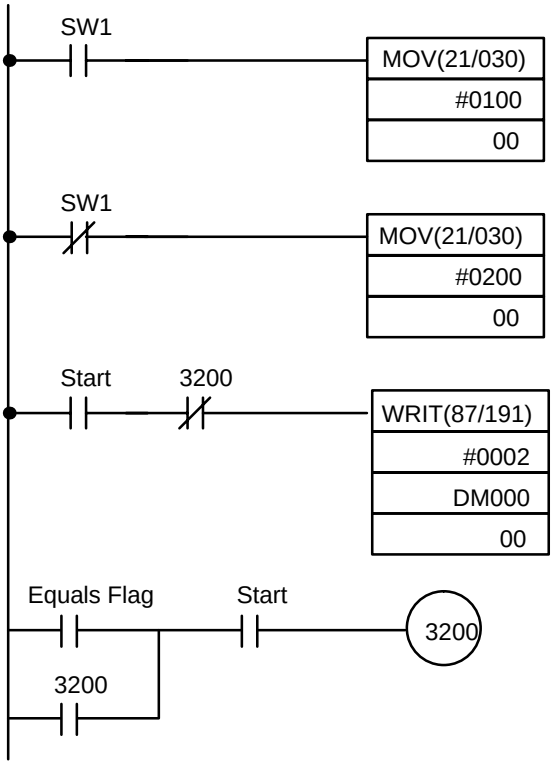
```
10  OPEN #2, "SCRN:(40)"
20  PC READ "B0"; R
30  RS$ = "ON"
40  IF R = 0 THEN  RS$ = "OFF"
50  PRINT #2, "RELAY = "; RS$
60  GOTO 20
```

#0001: Number of words to be transferred
DM000: First word to transfer (DM 000)
00: Destination word address

Example 8

Purpose: To retrieve and print several types of data from the PC using the PC GET statement and WRIT(87/191)

PC Program



ASCII Unit Program

```
10  OPEN #2, "LPRT : (47)"
20  PC READ "2I4" ; X, Y
30  PC GET I, J
40  IF J = 1 THEN GOTO 100
50  IF J = 2 THEN GOTO 200
60  GOTO 30
        .
        .
        .
100 PRINT #2, "DATA1 = " ;X
        .
        .
        .
200 PRINT #2, "DATA2 = " ;Y
```

Remarks:

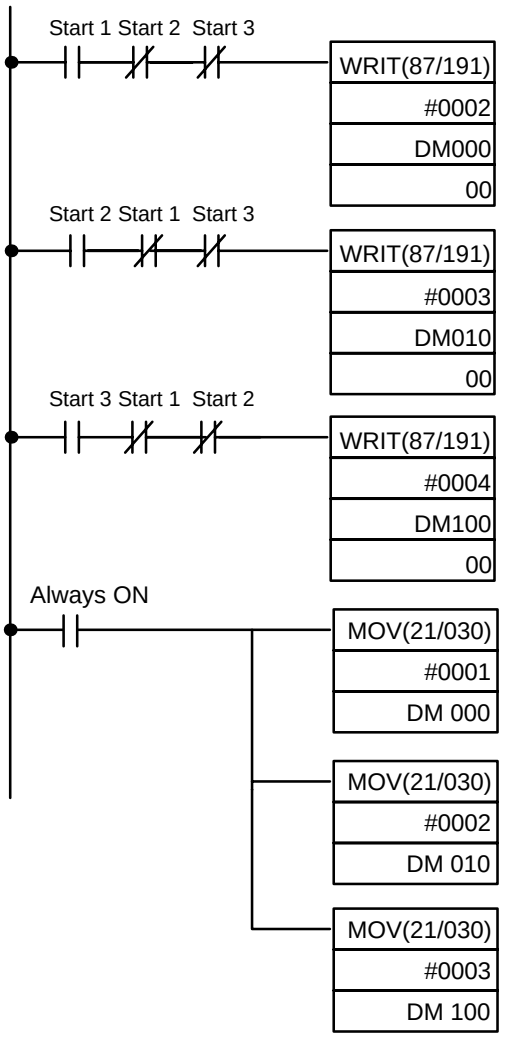
The two MOV(21/030) instructions place the data in the memory locations that will be read by the PC READ statement. After the MOV(21/030) instructions are executed, the Start flag is turned ON and WRIT(87/191) is executed.

Two lot size areas, stored in PC DM 000 and 001, are retrieved and printed.

Example 9

Purpose: To use PC interrupts to direct execution of the ASCII Unit

PC Program



ASCII Unit Program

```
 10  OPEN #2, “LPRT: (47)”
 20  ON PC 1 GOSUB 100
 30  ON PC 2 GOSUB 200
 40  ON PC 3 GOSUB 300
 50  PC ON
 60  GOTO 60

100  PC READ “2I4” ; X1, X2
110  PRINT #2, “DM1 = ”; X2
120  RETURN

200  PC READ “3I4” ; X1, X2, X3
210  PRINT #2, “DM11 = ” ; X2
220  PRINT #2, “DM12 = ” ; X3
230  RETURN

300  PC READ “4I4” ; X1, X2, X3, X4
310  PRINT #2, “DM101 = ” ; X2
320  PRINT #2, “DM102 = ” ; X3
330  PRINT #2, “DM103 = ” ; X4
340  RETURN
```

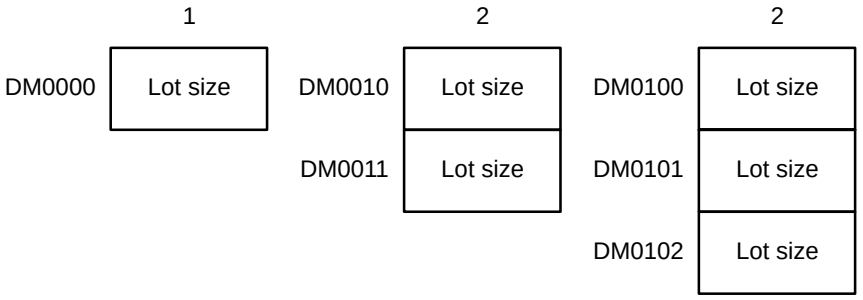
Remarks:

Three ON PC GOSUB statements are used to direct program execution to three different interrupt service routines. After the branch destinations are defined by the ON PC GOSUB statements, the ON PC statement is executed enabling the interrupts. The statement “GOTO 60” at line 60 causes the program to “sit and wait” for a PC interrupt to initiate further action.

If PC interrupt 1 interrupts the ASCII Unit, the contents of DM 000 will be printed. If PC interrupt 2 interrupts the ASCII Unit, the contents of DM 010 and 011 will be printed. If PC interrupt 3 interrupts the ASCII Unit, the contents of DM 100, 101, and 102 will be printed.

Connect the printer to port 2 and set the baud rate to 4,800 bps.

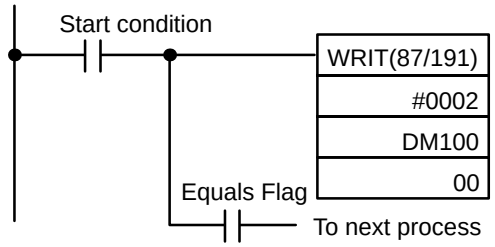
The lot sizes are stored in DM words as follows:



Example 10

Purpose: To print PC data and the time of data transfer

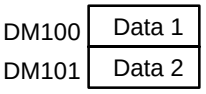
PC Program



ASCII Unit Program

```
10  OPEN #2, “LPRT: (47)”
20  C READ “2I4 ” ; D1, D2
30  PRINT #2, “DATA1 = ” ; D1,
    “DATA2 = ” ; D2, “TIME = ” ; TIME$
40  GOTO 20
```

#0002: Number of words to be transferred
DM100: First word to transfer (DM 000)
00: Destination word address



Remarks:

When the start condition is activated, PC data and the time of transfer are output to a printer connected to port 2 of the ASCII Unit. The PC read statement and WRIT(87/191) are used to obtain the data from the PC.

Output:

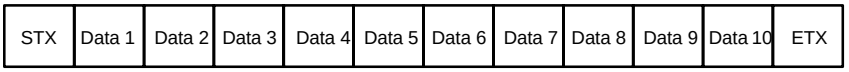
```
DATA1  =  5678        DATA2  =  9876        TIME = 13:45:03
DATA1  =  3249        DATA2  =  12          TIME = I4:02:51
```

Example 11

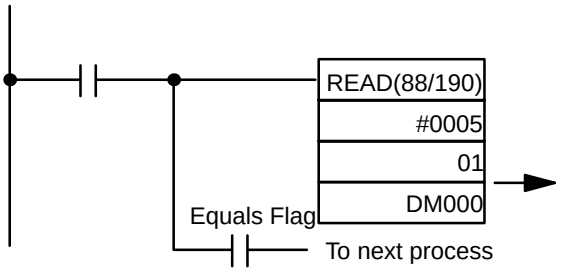
Purpose: To input data from a bar code reader using the PC WRITE statement

Remarks: Connect the bar code reader to port 2.

The following figure defines the output format of the bar code reader.



PC Program



ASCII Unit Program

```
10  OPEN #2, “COMU:(22)”
20  A$ = INPUT$ (1, #2)
30  IF A$ = CHR$(2) GOTO 50
40  GOTO 20
50  B$ = INPUT$(11, #2)
60  IF CHR$(3) = RIGHT$ (B$, 1)
    THEN B$ = MID$(B$, 1, 10)
    ELSE GOTO 20
70  PC WRITE “5A3” ; B$
80  GOTO 20
```

#0005: Number of words to be transferred

01: Destination word address

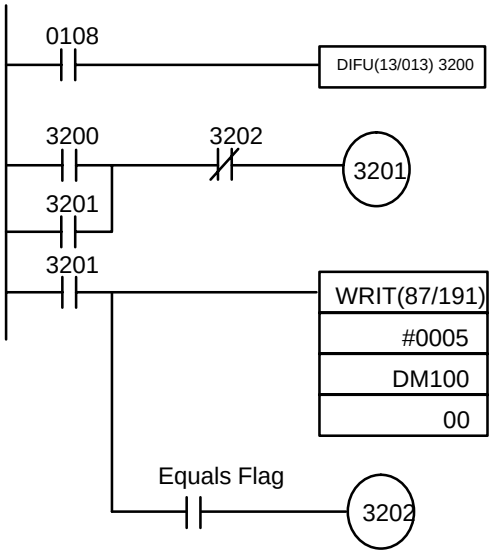
DM000: First word to transfer

DM000	Data 1	Data 2
DM001	Data 3	Data 4
DM002	Data 5	Data 6
DM003	Data 7	Data 8
DM004	Data 9	Data 10

Example 12

Purpose: To transfer data from the PC to the ASCII Unit with the ASCII Unit maintaining control

PC Program



ASCII Unit Program

```
100  PC PUT 1
110  PC READ “5I4” ; A1, A2, A3, A4, A5
120  PC PUT 0
130  PRINT A1, A2, A3, A4, A5
```

Execution Sequence:

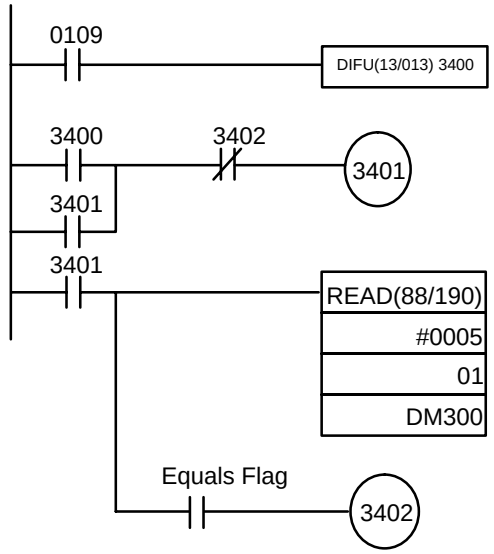
1. ASCII: The PC PUT 1 statement turns ON bit 0108
2. PC: The self-holding bit (3201) is set on the positive edge transition of bit 0108.
3. PC: WRIT(87/191) is executed.

4. PC: When execution of WRIT(87/191) is complete, the Equals Flag is turned ON and the self-holding bit is turned OFF.
5. ASCII: The data is read from the PC using PC READ
6. ASCII: Turns OFF bit 0108 using the PC PUT 0 statement
7. ASCII: Displays the data which is read in step 5.

Example 13

Purpose: To transfer data from the ASCII Unit to the PC with the ASCII Unit maintaining control

PC Program



ASCII Unit Program

```
100  PC PUT 2
110  PC WRITE "5I4" ; A1, A2, A3, A4, A5
120  FOR J = 1 TO 100 : NEXT J
130  PC PUT 0
```

Execution Sequence:

1. ASCII: Turns ON bit 0109 with the PC PUT 2 statement and executes the PC WRITE statement.
2. PC: The self-holding bit (3401) is set on the positive edge transition of bit 0109.
3. PC: Executes READ(88/190) when the self-holding bit (3401) is turned ON.
4. PC: Turns ON the Equals Flag after execution of READ(88/190) is completed and then turns OFF the self-holding bit (3401).
5. ASCII: Waits at line 120 until bit 0109 is turned ON by the PC. The wait time should be adjusted to the cycle time of the PC.
6. ASCII: Turns OFF bit 0109 with the PC PUT 0 statement.

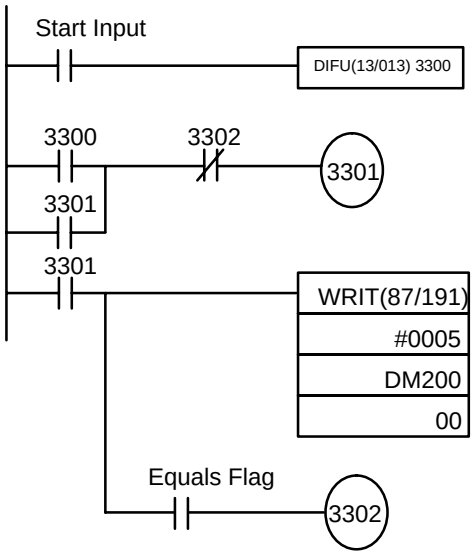
Remarks:

If the time required to transfer data from the ASCII Unit to the PC is shorter than one PC scan cycle, the PC cannot execute READ(88/190). In the above example, the ASCII Unit waits for the PC signal to be received at line 120.

Example 14

Purpose: To transfer data from the PC to the ASCII Unit with the PC maintaining control.

PC Program



ASCII Unit Program

```
 10  ON PC 1 GOSUB 100
 20  PC 1 ON
      ⋮
 90  GOTO 20
100  PC READ SUBROUTINE
110  PC READ "5I4" ; A1, A2, A3, A4, A5
120  PRINT A1, A2, A3, A4, A5
130  RETURN
```

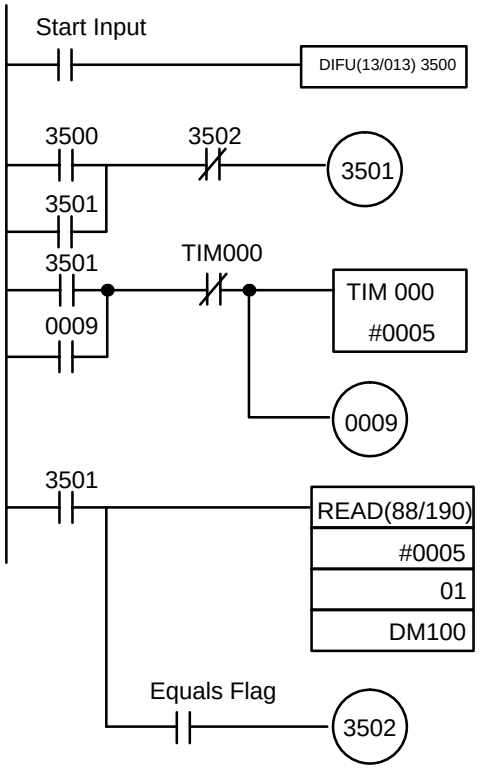
Execution Sequence:

1. PC: The self-holding bit (3301) is set on the leading edge of the start statement signal.
2. PC: WRIT(87/191) is executed.
3. PC: When execution of WRIT(87/191) is complete, the Equals Flag is turned ON and the self-holding bit is turned OFF.
4. ASCII: When the PC interrupts the ASCII Unit, execution branches to line 100 of the BASIC program and the data is read by the PC READ statement.
5. ASCII: Displays the data and processing returns to line 20 to await the next interrupt.

Example 15

Purpose: To transfer data from the ASCII Unit to the PC with the PC maintaining control.

PC Program



ASCII Unit Program

```
100   PC GET I, J
110   K = J AND 2
120   IF K <> 2 THEN GOTO 100
130   PC WRITE “5I4” ; A1, A2, A3, A4,
A5
```

Execution Sequence:

1. PC: The self-holding bit (3501) is set on the leading edge of the Start Input signal.
2. PC: Turns ON bit 0009 for 0.5 seconds with the TIM command after the self-holding bit has been turned ON.
3. PC: Executes READ(88/190)
4. PC: Turns ON the Equals Flag after READ(88/190) has been executed and then turns OFF the self-holding bit (3501).
5. ASCII: Monitors the setting of bit 0009 at lines 100 to 120.
6. ASCII: Executes the PC WRITE statement.

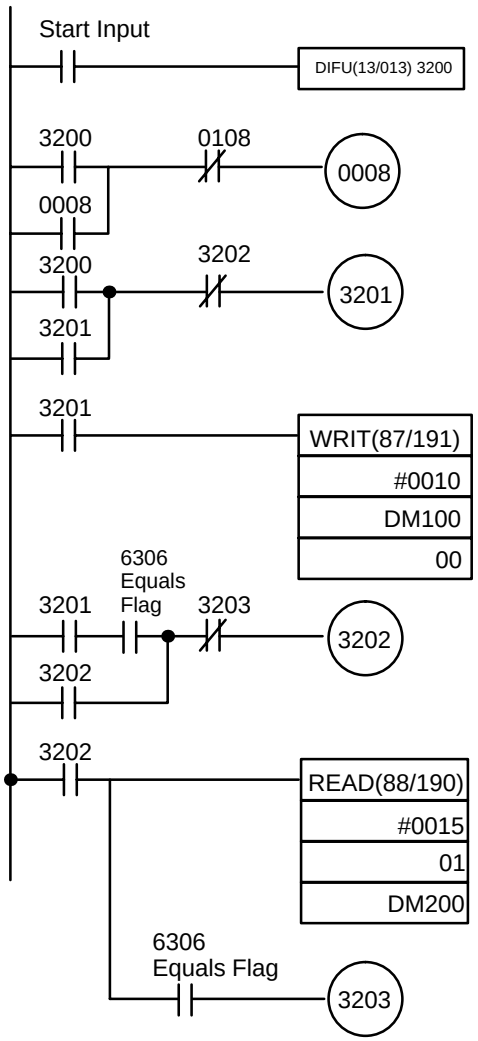
Example 16

Purpose: To process data with the ASCII Unit

Remarks:

This program transfers 10 words of data from the PC to the ASCII Unit (starting from PC DM 100) each time bit 1000 is turned ON. The ASCII Unit performs some calculations with the data and the results are sent back to the PC and stored in DM 200 to 214.

PC Program



ASCII Unit Program

```
100  PC GET J, K
110  L = K AND 1
120  IF L=1 THEN GOSUB 1000

     ( other processing )
990  GOTO 100
1000 SUBROUTINE
1010 PC PUT 1
1020 PC READ “10H4” ; A1, A2, . . . . , A10
1030 (computation processing: assigns the
     values to B1 through B10 )
1090
1100 PC WRITE “15H4” ; B1, B2, . . . , B15
1110 PC GET J, K
1120 L = K AND 1
1130 IF L = 0 THEN GOTO 1150
1140 GOTO 1110
1150 PC PUT 0
1160 RETURN
```

Execution Sequence:

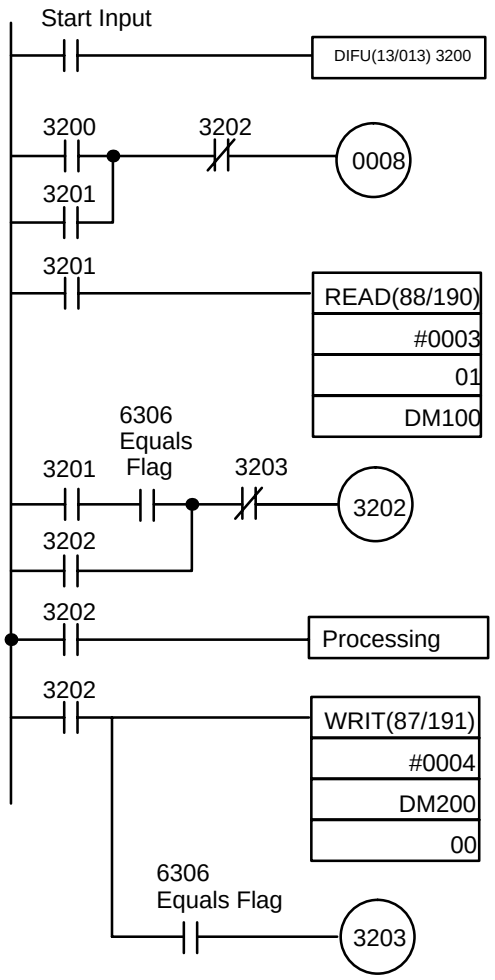
1. PC: Detects the positive edge transition of Start Input and turns ON bit 0008.
2. PC: Executes WRIT(87/191) when bit 3201 is turned ON.
3. PC: Turns ON the Equals Flag after the execution of WRIT(87/191) is completed.
4. ASCII: Reads the status of bit 0008 with the PC GET statement.
5. ASCII: If bit 0008 has been turned ON, execution branches to the subroutine beginning at line 1000.
6. ASCII: Turns ON bit 0108 with the PC PUT 1 statement at line 1010 and the self-holding bit (0008) is turned OFF.
7. ASCII: Executes the PC READ statement at line 1020 which assigns the PC data to variables A1 through A10.
8. ASCII: Performs computations on variables A1 through A10 and assigns the results to B1 through B15.
9. ASCII: Writes B1 through B15 to the PC at line 1100.
10. ASCII: Waits for bit 0008 to be cleared at lines 1110 through 1140.
11. ASCII: Turns OFF bit 0108 with the PC PUT 0 statement at line 1150.
12. PC: Transfers data written from the ASCII Unit to DM 200 through 214 using the READ(88/190).

13. PC: Turns ON the Equals Flag after execution of READ(88/190) has been completed and then turns OFF the self-holding bit.

Example 17

Purpose: To transfer data input through the ASCII Unit keyboard to the PC and then back to the ASCII Unit after computations have been performed by the PC.

PC Program



ASCII Unit Program

```
100  INPUT A1, A2, A3
110  PC PUT 1
120  PC WRITE “3I4” ; A1, A2, A3
130  PC READ “4I4” ; B1, B2, B3, B4
140  PC PUT 0
150  PRINT B1, B2, B3, B4
160  GOTO 100
```

Execution Sequence:

1. ASCII: Excepts input from the keyboard at line 100.
2. ASCII: Turns ON bit 0108 with the PC PUT 1 statement and then writes the entered data to the PC at line 120.
3. PC: Detects the positive-edge transition of bit 0108 and then turns ON the self-holding bit (3201).
4. PC: Executes READ(88/190) after bit 3201 is turned ON and reads data written from the ASCII Unit. The data is then transferred to DM 100 through 102.
5. PC: Turns ON the Equals Flag after execution of READ(88/190) has been completed. The self-holding bit (3201) is turned ON and the self-holding bit (3202) is turned OFF.
6. PC: Executes WRIT(87/191) after data processing has been completed and bit 3202 is turned ON. The data is then transferred to the ASCII Unit.

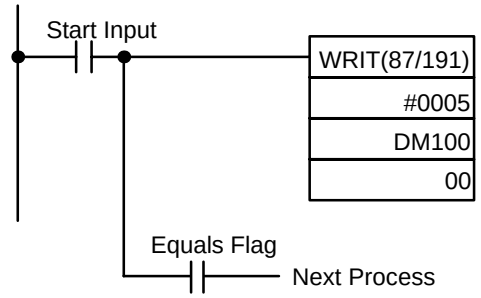
7. PC: Turns ON the Equals Flag when execution of WRIT(87/191) has been completed and then turns OFF the self-holding bit (3202).

8. ASCII: Data is read at line 130 and the results are assigned to the variables B1 through B4 and then displayed.

Example 18

Purpose: To initiate data transfer with the START switch using the WAIT statement

PC Program



ASCII Unit Program

```
100   PRINT “START”
110   WAIT “10:00.0” , 1000
120   PC READ “I4” ; A, B, C, D, E
130   PRINT A, B, C, D, E
I40   END
1000  PRINT “ERROR READY? Y/N”
1010  F$ = INKEY$
1020  IF F$ = “Y” THEN 100
1030  IF F$ = “N” THEN END
ELSE 1010
```

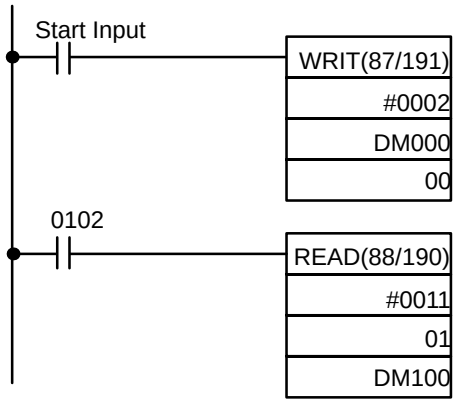
Remarks:

Pressing the PC START switch will cause specified PC data to be transferred to the ASCII Unit and displayed on the monitor. When the program is executed the message “Ready” will be displayed on the screen. If the START switch is not pressed within ten minutes, an error message will be displayed.

Example 19

Purpose: To direct processing using different interrupts

PC Program



ASCII Unit Program

```
 10  OPEN #1, “TERM:(42)”
 20  OPEN #2, “COMU:(42)”
 30  ON KEY 1 GOTO 100
 40  ON KEY 2 GOTO 200
 50  ON PC GOSUB 300
 60  ON COM2 GOSUB 400
 70  KEY ON: COM2 STOP
 80  GOTO 80
100  ’ KEY 1 PROCESSING
110  COM2 ON: PC ON
120  GOTO 120
200  ’ KEY 2 PROCESSING
210  COM2 ON
220  IF A  =  1 THEN GOSUB 300
230  GOTO 220
300  ’ PC INTERRUPT PROCESSING
310  PC READ “2I4” ; P, Q
320  LENG = LEN(A$)
330  PCWRITE “I4, 10A3” ; LENG, A$
340  A  =  0
350  RETURN
400  ’COM INTERRUPT PROCESSING
410  IF EOF(2) THEN RETURN
420  A$  =  INPUT$ (LOC(2), #2)
430  A  =  1
440  RETURN
```

Remarks:

In this example, a terminal is connected to port 1 and an RS-232C communication device is connected to port 2. Initially, all the interrupts are disabled. The program will wait for one of two inputs from the keyboard — KEY 1 or KEY 2, each of which will direct the program to process subsequent interrupts in a unique way.

1. If key 1 is pressed, the COM2 and PC interrupts will be enabled. When COM2 interrupts the ASCII Unit, a character is read from the communication device and assigned to the variable A$. When the PC subsequently interrupts the ASCII Unit, the character will be written to the PC.
2. If key 2 is pressed, only the COM 2 interrupt is enabled. When COM 2 interrupts the ASCII Unit, the data is read and written directly to the PC.

6-3 Programs in Four-word Mode

This section presents example programs with the ASCII Unit set in four-word mode. There are also several ASCII programs that do not require a PC program in order to run.

For all of the following examples:

- printer is connected to port 2
- 8 bits/ no parity/ 2 stop bits

Example 1

Purpose: To print data at fixed time intervals using the LPRINT statement

This example does not require a PC data transfer routine.

ASCII Unit Program:

```
100     TH$ = MID$(TIME$,1,2)
110     IF TH$ = TH0$ GOTO 200
120     TH0$ = TH$
130     LPRINT TIME$,A
```

Remarks:

This program example prints a value (A) and the present time (TIME$), on a printer, every hour, on the hour. The PRINT statement is executed when the “hours” change on the internal clock (for example, when the time changes from 9:59 to 10:00). The clock (24-hour) must be set prior to program execution.

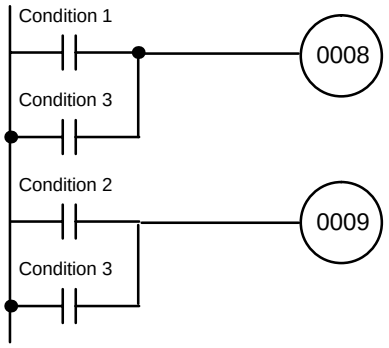
Example 2

Purpose: To direct execution of the ASCII Unit from the PC using the PC GET statement

Another way to externally control program execution is through poling. Poling is the process of continuously checking the value of a specified bit or word. If the value of the bit or word matches a condition set in the program, a corresponding branch instruction is executed.

In the following program, the ASCII Unit PC GET statement is used to pole a specific word of the PC.

PC Program



ASCII Unit Program

```
10    PC GET I, J
20    K = J AND 3
30    IF K = 1 GOTO 100
40    IF K = 2 GOTO 200
50    IF K = 3 GOTO 300
60    GOTO 10
```

Remarks:

The PC GET statement reads bits 10008 to 10015 of the PC as a word. The word is logically ANDed with 3 (00000011) and the result of this operation is used to branch the program. When bit 10008 is turned ON, k will be equal to 1 and the program will branch to line 100. If bit 10009 is turned ON, k will be equal to 2 and the program will branch to line 200.

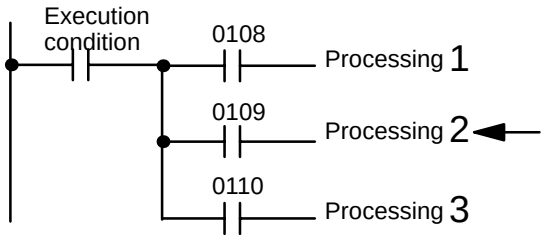
Example 3

Purpose: To control execution of the PC from the ASCII Unit using the PC PUT statement

Remarks:

Using the PC PUT statement, the ASCII Unit can write data to bits 08 to 15 of word n+3 of the PC. If the value of this data matches a condition set in the PC program, a corresponding branch instruction will be executed.

PC Program



ASCII Unit Program

```
10    OPEN #2, "KYBD:"
20    INPUT #2, A
30    PC PUT A
```

Remarks:

In the above program, the ASCII Unit accepts external input from a keyboard using the INPUT statement and transfers that data to the PC with the PC PUT statement.

If the number "1" is input by the device which is connected to port #2 (A=1), bit 0108 of the PC is turned ON, allowing process 1 to be executed.

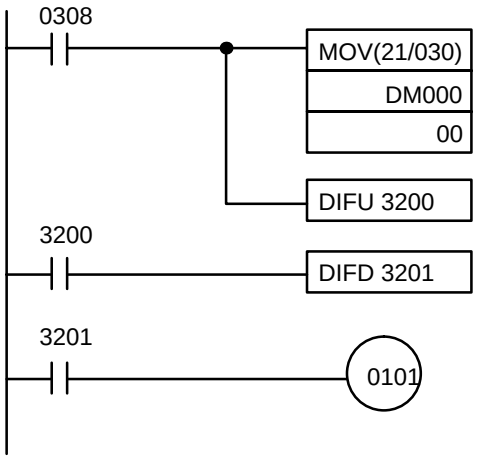
If the number "2" is input by the device which is connected to port #2 (A=2), bit 0109 of the PC is turned ON, allowing process 2 to be executed.

If the number "3" is input by the device which is connected to port #2 (A=3), bit 0110 of the PC is turned ON, allowing process 3 to be executed.

Example 4

Purpose: To print out production data every hour from DM000.

PC Program



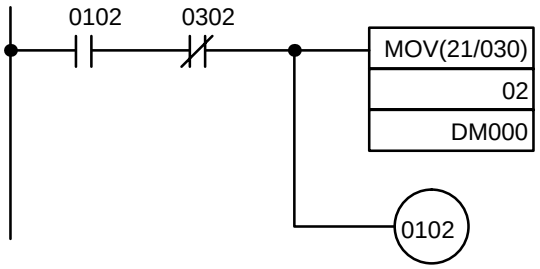
ASCII Unit Program

```
10    ' DM PRINTING PROGRAM
20    OPEN #2, "LPRT: (47)"
30    A$ = 00 : 00"
40    B$ = MID$(TIME$, 4, 5)
50    IF B$ <> A$ GOTO 40
60    PC PUT 1
70    PC READ "I4" ; X
80    PRINT #2, "DM = " ; X
190   PC PUT 0
100   GOTO 40
```

Example 5

Purpose: To accept input from the keyboard and write it to the PC using the PC WRITE statement

PC Program



ASCII Unit Program

```
10    INPUT I
20    PC WRITE "I4" ; I
30    GOTO 10
```

Remarks:

Product codes stored in DM memory are replaced by data input through a keyboard. The data is represented as 4-digit hexadecimal numbers.

Example 6

Purpose: To read data from an input file through a communications port

ASCII Unit Program

```
 10    CLEAR 1000
100    OPEN #1,"COMU:"
110    OPEN #2,"COMU:"
120    ON COM1 GOSUB 1000
130    ON COM2 GOSUB 2000
140    COM1 ON:COM2 ON
150    GOTO 150
1000   A = LOC(1)
1010   IF A<>0 THEN
       A$ = A$+INPUT$(A,#1)
1020   RETURN
```

```
2000    B = LOC(2)
2010    IF B<>0 THEN
            B$ = B$+INPUT$(B,#2)
2020    RETURN
```

Example 7

Purpose: To transfer multi-word data from the ASCII Unit to the PC in four-word mode by using the PC WRITE statement continuously.

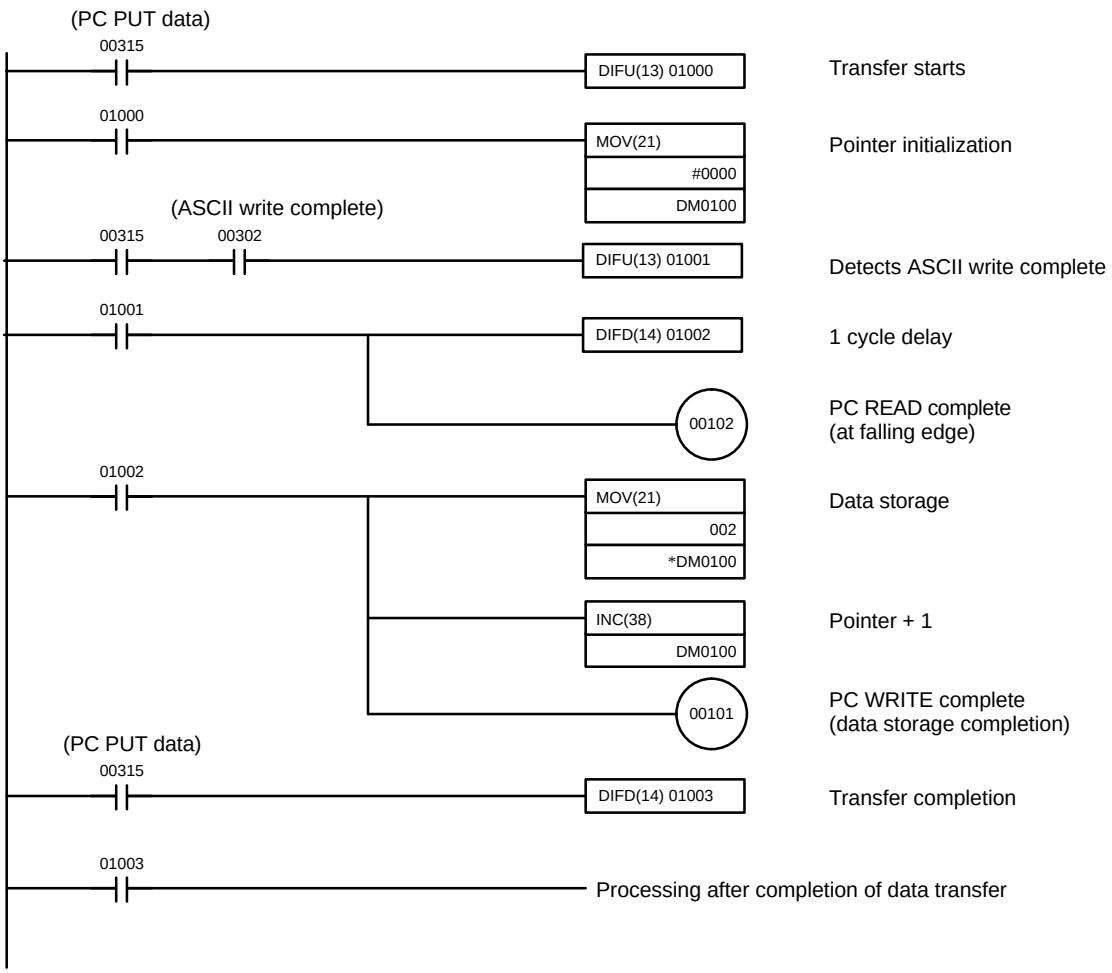
1, 2, 3...

1. Data transfer from the ASCII Unit starts when bit 15 of the Wd (n+3) is turned ON with PC PUT.
2. The length of the first parameter is transferred to the PC.
3. The given ASCII data of the parameter is transferred to the PC in groups of two characters.
4. If the number of data items is an odd number, * is added to the end of the data before transfer.
5. Data transfer from the ASCII Unit is complete when bit 15 of the Wd (n+3) is turned OFF with PC PUT.

ASCII Unit Program

```
100 '  *****  Writes word-by-word to the PC when four-word mode is set on the ASCII Unit.  *****
110 '  *****  The number of characters and the character string of A$ are transferred to the  *****
120 '  *****  PC in groups of two characters. If the number of characters is an odd number  *****
130 '  *****   "*" will be added.                                                           *****
140 '  *****  A$="1234567" ——>  DM0000 0007  The length of the number of characters  *****
150 '  *****                    DM0001 3132  Data 1 and 2                           *****
160 '  *****                    DM0002 3334  Data 3 and 4                           *****
170 '  *****                    DM0003 3536  Data 5 and 6                           *****
180 '  *****                    DM0004 372A  Data 7. If an odd number, *(2A) added  *****
 190    A$="1234567890ABCDEFGHIJK"      'Data transferred.
 200    L=LEN(A$)                       'Calculation of the number of characters.
 210    IF L MOD 2 = 1 THEN A$=A$+"*"   'Make the number of characters to be an even number.
 220    M%=L/2                          'Round M to the nearest whole number.
 230    PC PUT 128                      'Turn ON bit 15 of Wd(n+3) after transfer starts.
 240        PC WRITE "I4";L             'PC WRITE the number of characters.
 250        PC READ "H4";X              'Confirm the completion on the PC side.
 260        FOR T=1 TO M%
 270            B$=MID$(A$, T*2–1, 2)   'Take out in groups of two characters.
 280            PC WRITE "A3";B$        'Write to the PC.
 290            PC READ "H4";X          'Confirm completion on the PC side.
 300        NEXT T
 310    PC PUT 0                        'Turn OFF bit 15 of Wd(n+3) on transfer completion.
 320    END
```

PC Program

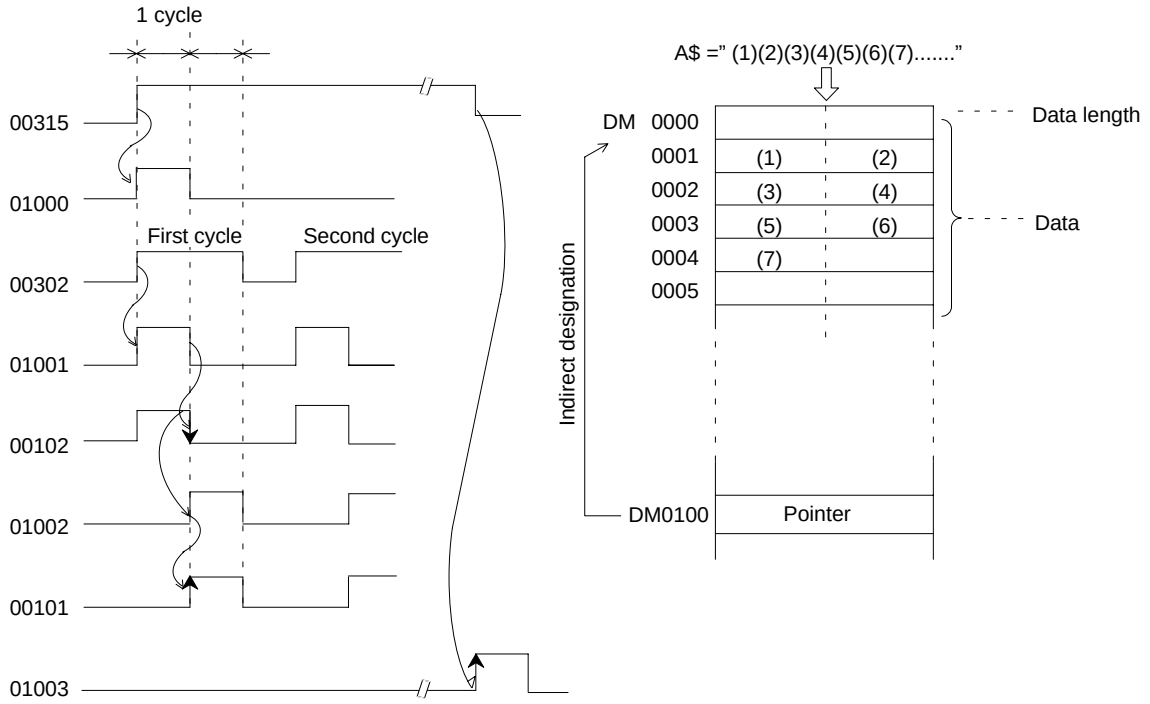


Remarks:

After 00315 is ON, store data from the ASCII Unit in sequence from DM 0000 onward.

On PC WRITE completion, the ASCII Unit will be informed that the data has been stored.

Transfer completion is detected when 00315 is OFF.



Example 8

Purpose: To transfer multi-word data from the PC to the ASCII Unit in four-word mode by using the PC READ statement continuously.

1, 2, 3...

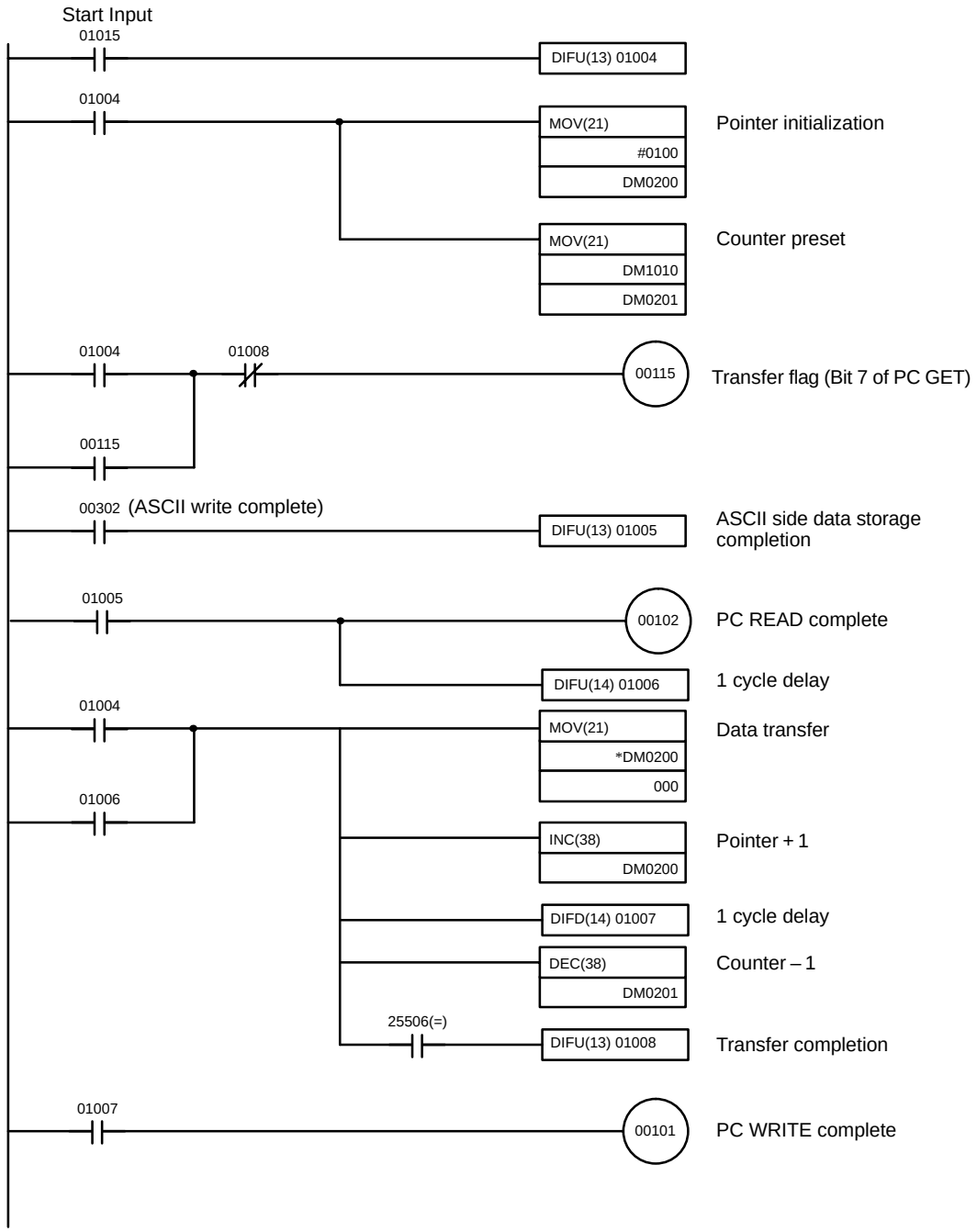
1. Data transfer from the PC starts when the PC turns ON bit 15 of Wd(n+1), and continues until the program reaches PC GET.
2. When bit 15 of Wd(n+1) is turned ON, PC READ is executed and the character string is continued.
3. The PC is informed of PC READ completion by PC WRITE. From word 2 on, PC READ and PC WRITE are repeated until bit 15 of Wd(n+1) is turned OFF.
4. Transfer is complete when bit 15 of Wd(n+1) is turned OFF.

ASCII Unit Program

```
800 '   *****                                                              *****
810 '   *****  Reads word-by-word from the PC when four-word mode is set on the   *****
820 '   *****  ASCII Unit. Data read is stored in A$.                      *****
830 '   *****       DM0100 0004   The length of the number of characters   *****
840 '   *****       DM0101 3132   Data 1 and 2 ——> A$="1234ABCD"           *****
850 '   *****       DM0102 3334   Data 3 and 4                             *****
860 '   *****       DM0103 4142   Data 5 and 6                             *****
870 '   *****       DM0104 4344   Data 7 and 8                             *****
880 '   *****                                                              *****
 890    A$=" "                          'Initialization of the parameters to be stored.
 900    PC GET H, I                     'Check bit 15 of Wd(n+1).
 910    IF I AND 128 <> 128 GOTO 900    'Transfer start?
 920        PC READ "A3"; B$            'One word read.
 930        PC WRITE "A3"; B$           'Inform the PC of the completion of READ.
 940        A$=A$+B$                    'Edit read data.
```

```
950     PC GET H, I                         'Check bit 15 of Wd(n+1).
960     IF I AND 128 = 128 GOTO 920         'Completion of transfer?
970     END
```

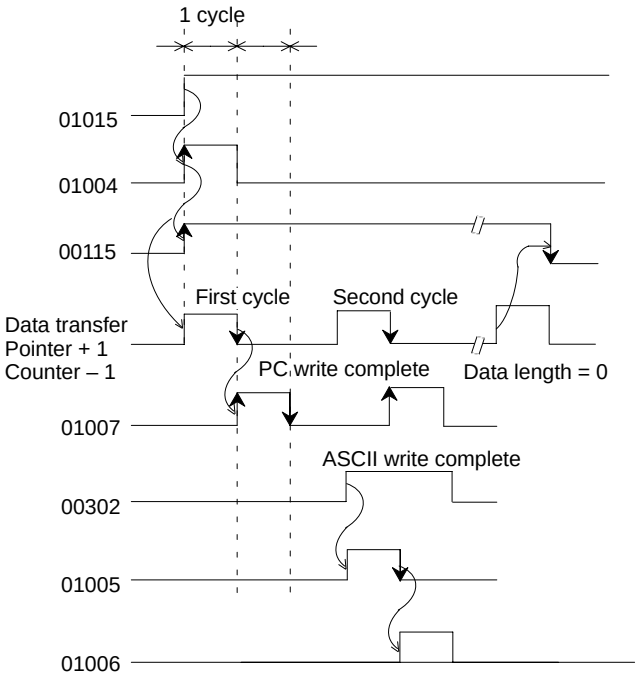
PC Program

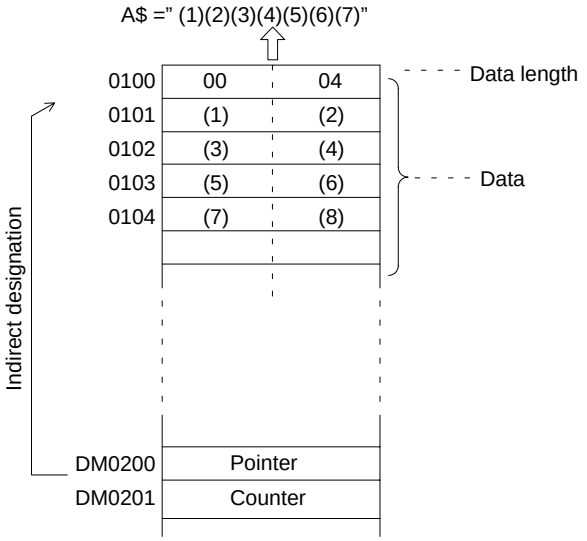


Remarks:

After Start Input is ON, the PC transfers the data from DM 0101 onward based on the contents of DM 0100 as the data length. From word 2 on, the data is transferred whenever the ASCII write complete instruction is ON.

When data transfer starts, 00115 is turned ON and when data transfer is completed, 00115 is turned OFF to inform the ASCII Unit of data transfer and completion.





6-4 Assembly Language Examples

Example 1: Classification of Characters

This program divides characters that are input from the keyboard into numeric and character strings and then recombines them.

BASIC Program

```
100     DEF USR0=&H2000
110     INPUT A$
120     A$=USR0(A$)
130     PRINT A$
140     END
```

1, 2, 3...

1. Use MSET &H3000 to reserves an assembly language program area.
2. Key in MON to initiate assembly language monitor mode.
3. Key in CTRL+A <- Sets mini-assembler mode.
4. Key in the program sequentially from $2000.
5. Key in CTRL+B after the program has been input to return to BASIC mode.

The following memory areas are used as a program area, work area, and buffer area:

Program Area

$2000 to $24FF	Program area

Work Area

$2500 to $2501	Stores buffer 1 (stores numerals) pointer
$2502 to $2503	Stores buffer 2 (stores characters) pointer
$2504 to $2505	Stores transfer source word
$2506 to $2507	Stores transfer destination word

Buffer Area

$25600 to $26FF	Numeral storage area
$2700 to $27FF	Character storage area

Assembly Program

Assembly language program operation:

The numbers and characters are separated and stored in the number storage buffer and the character storage buffer, respectively. Then numeric strings and character strings are restored as the original character variables. This program has no practical application; it's just an example.

```
$2000  PSHA   Saves registers
       PSHB
       PSHX
       LDD    #$2600  Sets first address of buffer 1 in point 1
       STD    $2500
       LDD    #$2700  Sets first address of buffer 2 in point 2
       STD    $2502
       LDAB   0,X     Number of characters to GET
       LDX    1,XC    Character variable first address GET
       STX    $2504
$2016  LDX    $2504   DOUNTIL (number of times equal to the number of characters)
       LDAA   0,X     Character GET
       INX    Character variable address pointer +1
       STX    $2504
       CMPA   #$30    IF (minimum $30)
       BLT    $2032   THEN
       CMPA   #$39    IF (numeral less than $39)
       BHI    $2032   THEN
       LDX    $2500   Stores numeral in buffer 1
       STAA   0,X
       INX
       STX    $2500
       BRA    $203B
$2032  LDX    $2502   ENDIF
       STAA   0,X     Stores character in buffer 2
       INX
       STX    $2502
$203B  DECB   Updates counter
       BNE    $2016   ENDDO
       LDD    $2500
       LDX    #$2600  Transfer from buffer 1 to a character variable
       STX    $2504
       SUBD   #$2600
       PULX
       PSHX
       PSHB
       LDX    1,X
       STX    $2506
       JSR    $2100
       LDX    #$2700  Transfer from buffer 2 to a character variable
       STX    $2504
       PULB
       PULX
       PSHX
       LDX    1,X
       ABX
       STX    $2506
       LDD    $2502
       SUBD   #$2700
       JSR    $2100
       PULX
       PULB
       PULA
       RTS
$2100  LDX    $2504   Data transfer subroutine
       LDAA   0,X
       INX
       STX    $2504
       LDX    $2506
       STAA   0,X
       INX
       STX    $2506
       DECB
       BNE    $2100
       RTS
```

Example 2: Use of More than One Parameter

This program singles out the larger of two character strings.

Three parameters are used (i.e., the two original character strings for comparison and the other for result storage).

BASIC Program

```
100 '  *****                                                  *****  Assembly language
110 '  ***** Program to single out the larger of two character  *****  program $2170 to
120 '  ***** strings                                          *****  $21AF. Work area
```

```
130 '   *****                                          *****   $2000 to $2005.
140     CDX$="13426285903581693417"   'Original character string for comparison CDX$.
150     CDY$="57201674337291551930"   'Original character string for comparison CDY$.
160     ANS$="00000000000000000000"   'Result storage character string ANS$.
170     DEF USR0=&H2170               'Storage address definition of assembly language function.
180     CX%=VARPTR(CDY$)+1            'Calculation of the storage address of CDX$.
190     POKE &H2000, CX% \ 256        'Leftmost storage address of CDX$ --> Work area
200     POKE &H2001, CX% MOD 256      'Rightmost storage address of CDX$ --> Work area
210     CY%=VARPTR(CDY$)+1            'Calculation of the storage address of CDY$.
220     POKE &H2002, CY% / 256        'Leftmost storage address of CDY$ --> Work area
230     POKE &H2003, CY% MOD 256      'Rightmost storage address of CDY$ --> Work area
240     ANS$=USR0(ANS$)               'Execute assembly language function.
250     PRINT ANS$
260     END
```

Operation

Use VARPTR to obtain the addresses of parameters to be used in the assembly language function program and store them in the work area in advance. In the above example, three parameters are used in the assembly language function program.

Note The addresses of parameters are calculated as integral parameters.

Parameters with the parameter name format "XXXX" will not be stored in the parameter area but the data in the source program will be used. Therefore, after the execution of line 240, the value between the quotation marks in line 160 will change.

Assembly Program

```
2170  E8  00      LDAB  $00, X   'The length of ANS$ → B register
2172  EE  01      LDX   $01, X
2174  FF  20 04   STX   $2004    'ANS$ address memory.
2177  FF  20 00   LDX   $2000
217A  EE  00      LDX   $00, X
217C  FF  20 00   STX   $2000    'SDX$ address memory.
217F  FE  20 02   LDX   $2002
2182  EE  00      LDX   $00, X
2184  FF  20 02   STX   $2002    'SDY$ address memory.
2187  FE  20 00   LDX   $2000
218A  A6  00      LDAA  $00, X   'SDX$ data read.
218C  08          INX
218D  FF  20 00   STX   $2000
2190  FE  20 02   LDX   $2002
2193  A1  00      CMPA  $00, X   'Comparison with the data of SDY$.
2195  24  02      BCC   $2199    'The data of SDX$ < the data of SDY$?
2197  A6  00      LDAA  $00, X   'SDY$ data read.
2199  08          INX
219A  FF  20 02   STX   $2002
219D  FE  20 04   LDX   $2004
```

```
21A0   A7   00       STAA    $00, X    'Writes the larger character string to ANS$.
21A2   08             INX
21A3   FF   20   04   STX     $2004
21A6   5A             DECB
21A7   26   DE       BNE     $2187     'Complete?
21A9   39             RTS
```

Work Area

$2000 $2001		Storage address memory area of parameter SDX$.
$2002 $2003		Storage address memory area of parameter SDY$.
$2004 $2005		Storage address memory area of parameter ANS$.

Example 3: FCS Calculation

This program calculates the FCS to be used in the host link assembly language.

Character strings to be calculated are DA$ and character strings in which calculation results are stored as FCS$.

BASIC Program (100 to 230 Lines)

```
100 '   *****  FCS calculation (using          *****
 110'   ***** assembly language function)      *****   Assembly language program $2100 to $213F.
120 '   ***** Calculate the FCS of DA$ and     *****   Work area $2000 to $2001.
130 '   ***** obtain the result as ANS$        *****
 140    DA$="@10RR00310123"                    'The DATA string to calculate the FCS.
 150    FCS$="43"                              'FCS storage character string. (The contents of "XX" will be
                                               substituted and converted when the program runs.)
 160    DEFUSR0=&H2100                         'Storage address definition of assembly language function
 170    B%=VARPTR(FCS$)+1                      'Calculation of the storage address of FCS$.
 180    POKE &H2000, B% \ 256                  'Leftmost storage address of FCS$ --> Work area
 190    POKE &2001, B% MOD 256                 'Rightmost storage address of FCS$ --> Work area
 200    DA$=USR0(DA$)                          'Execute assembly language function.
 210    ANS$=DA$+FCS$                          'Create a character string added with the FCS.
 220    PRINT ANS$
 230    END
240 '
500 '   ***** FCS calculations (BASIC          *****
510 '   ***** instructions only)               *****
520 '   *****Calculate the FCS of DA$ and      *****
530 '   ***** obtain the result as ANS$        *****
 540    DA$="@10RR00310123"
 550    L=LEN(DA$)
 560    Q=0
 570    FOR N=1 TO L
 580    Q=ASC(MID$(DA$, N, 1)) XOR Q
 590    NEXT
 600    FCS$=HEX$(Q)
```

```
610  IF LEN(FCS$)=1 THEN FCS$="0"+FCS$
620  ANS$=DA$+FCS$
630  PRINT ANS$
640  END
```

As seen above, there is a program which is calculated using BASIC instructions in lines 500 to 640 for purposes of comparison.

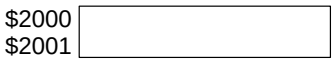
The execution times required by the assembly language functions and BASIC instructions are as follows:

Assembly language functions (lines 140 to 220): 29 ms
BASIC instructions (lines 540 to 630): 160 ms

Assembly Program

```
2100  E6  00      LDAB   $00, X   'The length of DA$ → B register
2102  EE  01      LDX    $01, X   'The storage of DA$ → X register
2104  4F          CLRA
2105  A8  00      EORA   $00, X   'Calculate the EOR.
2107  08          INX
2108  5A          DECB
2109  26  FA      BNE    $2105    'Repeat for the number of character strings.
210B  16          TAB
210C  C4  0F      ANDB   #$0F     'ASCII conversion of the FCS value.
210E  C1  0A      CMPB   #$0A
2110  25  02      BCS    $2114    'If the rightmost digit of the FCS ≧ 10
2112  CB  09      ADDB   #$07     ' THEN convert to A to F.
2114  CB  30      ADDB   #$30
2116  44          LSRA
2117  44          LSRA
2118  44          LSRA
2119  44          LSRA
211A  81  0A      CMPA   #$0A
211C  25  02      BCS    $2120    'If the leftmost digit of the FCS ≧ 10
211E  8B  09      ADDA   #$07     ' THEN convert to A to F.
2120  8B  30      ADDA   #$30
2122  FE  20  00  LDX    $2000
2125  EE  00      LDX    $00, X
2127  ED  00      STD    $00, X   'Store the data in the FCS$ area.
2129  39          RTS
```

Work Area



Storage address memory area of parameter FCS$.

Note The address of parameter FCS$ is stored in $2000 and $2001 before retrieving the assembly program.

Appendix A
Standard Models

Item	Description	Model No.
ASCII Unit	EEPROM	C500-ASC04
Battery Set	Backup battery for C500 only	C500-BAT08

Appendix B
Specifications

Item	Specifications
Communication mode	Half duplex
Synchronization	Start-stop
Baud rate	Port 1: 300/600/1,200/2,400/4,800/9,600 bps Port 2: 300/600/1,200/2,400/4,800/9,600/19,200 bps (switch selectable)
Transmission mode	Point-to-point
Transmission distance	15 m max.
Interface	Conforms to RS-232C. Two ports (D-sub 25P connectors)
Memory capacity	BASIC program area and BASIC data area: 24K bytes (RAM) (memory is protected by built-in battery backup) BASIC program storage area: 24K bytes (EEPROM) The program memory area can be segmented into 3 individual program areas.
Transfer capacity	255 words at a maximum of 20 words per cycle
Timer function	Year, month, day, date, hour, minute, second (leap year can be programmed) Accuracy: month + 30 seconds (at 25°C)
Diagnostic functions	CPU watchdog timer, battery voltage drop
Battery life	5 years at 25°C. (The life of the battery is shortened if the ASCII Unit is used at higher temperatures.)
Internal current consumption	200 mA max. at 5 VDC
Dimensions	34.5 x 250 x 93 (HxWxD) mm
Weight	300 grams max.
EEPROM	Has a lifetime of 5,000 saves

Note Abnormal data may be output on the ports when power is turned ON. Set up the device receiving the data to ignore (e.g., clear) any abnormal data output during startup procedures.

Front Panel DIP Switch

Pin No.	Function	Description
1	Start mode	Sets automatic or manual mode for power-on start-up of a BASIC program.
2	Automatic program transfer from EEPROM to RAM	Specifies whether the BASIC program is automatically transferred from the EEPROM to RAM on power application or reset
3 and 4	Program No.	This pin sets the program number. The program number can be changed by the PGEN command.
5	Data Section mode selector	This pin sets the Data Section to either two-word or four-word mode
6 to 8	Screen size	Sets the screen size of the of the input device

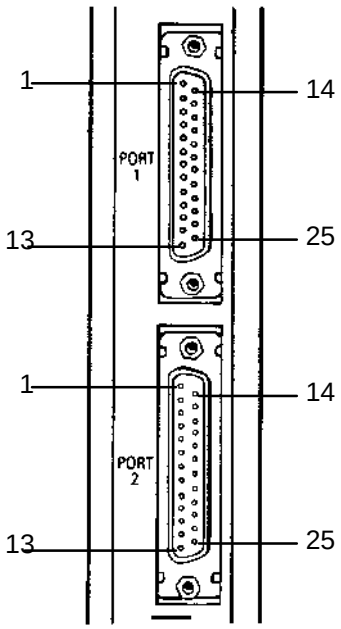
Back Panel DIP Switch

Pin No.	Function	Description
1 to 3	Baud rate for Port 1	Sets the baud rate for Port 1.
4 to 6	Baud rate for Port 2	Sets the baud rate for Port 2.
7 and 8	Not used	Always set these pins to OFF.

RS-232 Interface

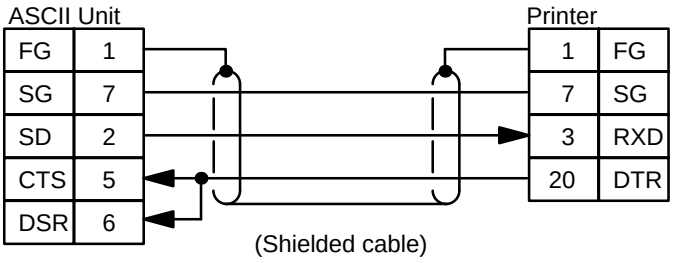
The ASCII Unit is connected to peripheral devices through two RS-232C interfaces. To connect peripheral devices to the ASCII Unit, use the included connectors.

The following figure shows the RS-232C connectors on the ASCII Unit. The electrical characteristics of these connectors conform to the EIA-RS-232C standards. Signal directions are oriented from the point of view of the ASCII Unit.

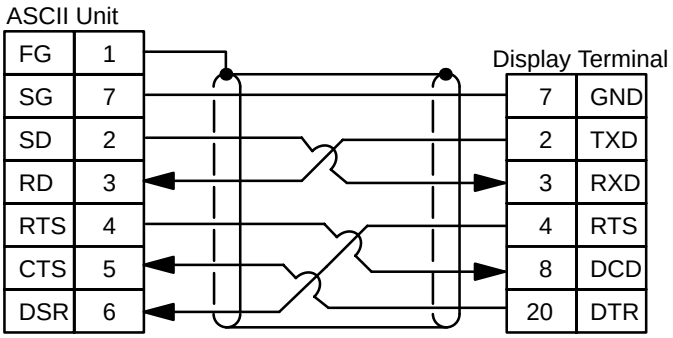


Pin No.	Symbol	Name	Direction
1	FG	Frame ground	
2	SD	Send data	Output
3	RD	Receive data	Input
4	RTS	Request to send	Output
5	CTS	Clear to send	Input
6	DSR	Data send ready	Input
7	SG	Signal ground	
8 to 19	---	Not used	
20	DTR	Data terminal ready	Output
21 to 25	---	Not used	

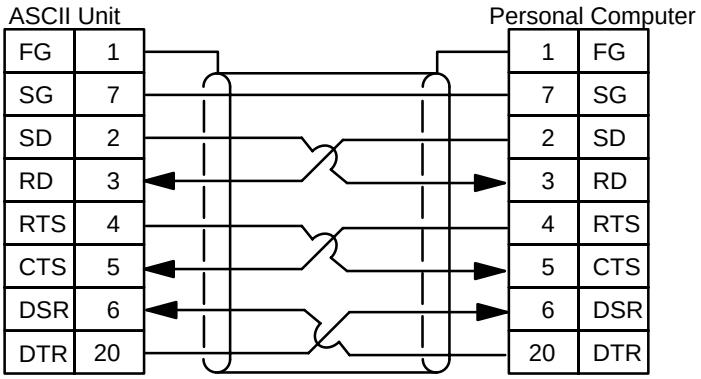
Connections to Peripheral Devices



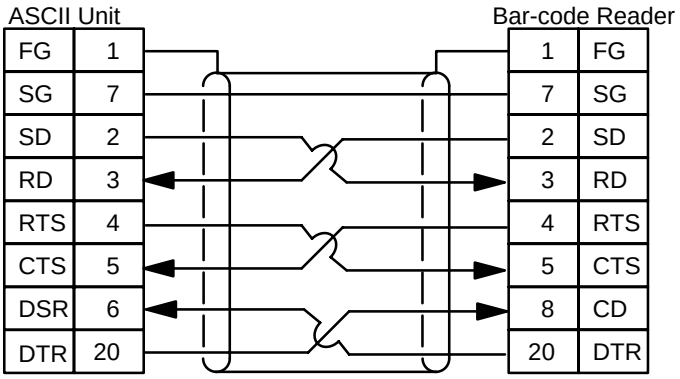
(Shielded cable)



(Shielded cable)



(Shielded cable)



(Shielded cable)

Interface Signal Timing

Before using any port after the ASCII Unit is turned on or restarted, Port 1 is assigned to the peripheral device TERM and Port 2 is assigned to LPRT. When there is an input or output at a port, the RTS, STS, DTR, and DSR signals are treated as described below.

Transmission from the ASCII Unit to a Peripheral Device

The default setting of the DTR signal is ON at Port 1 and OFF at Port 2. When the OPEN instruction is executed, the condition of the DTR signal varies with the peripheral device as follows:

Peripheral device	TERM	SCRN	KEYB	COMU	LRPT
Condition of DTR	ON	OFF	ON	ON	OFF

Note ON: HIGH
OFF: LOW

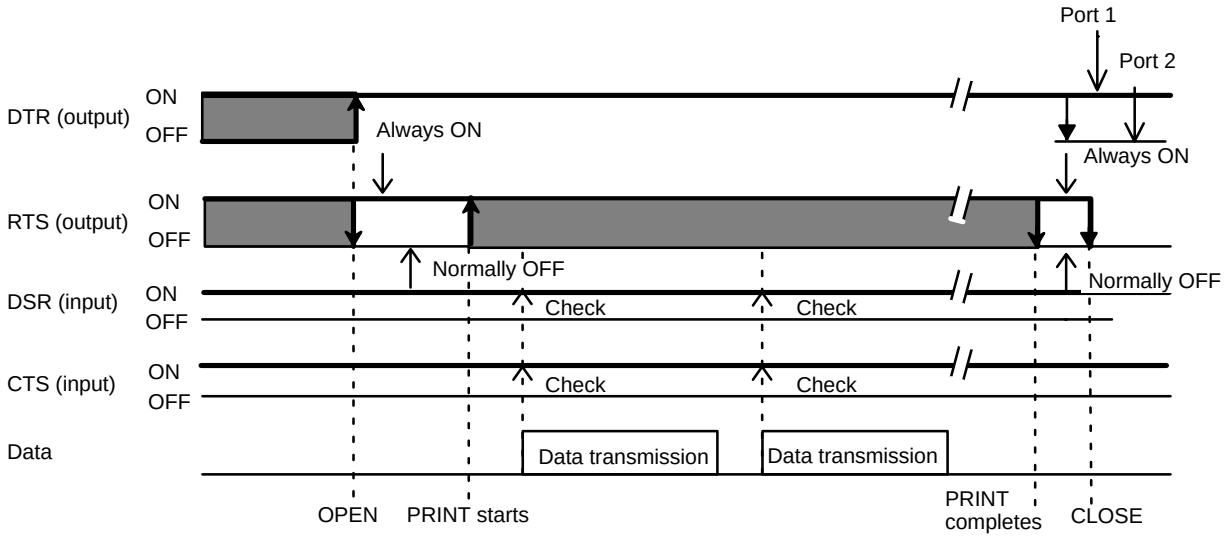
The RTS signal will be OFF if the effective signal wire is X (normally OFF) when the OPEN instruction is executed. If the effective signal wire is O (normally ON), the RTS signal will be ON from the execution of the OPEN instruction until the execution of the CLOSE instruction.

When the PRINT instruction is executed, the RTS signal will be ON and the ASCII Unit will transmit data after confirming that the CTS and DSR signals are both ON. If these signals are not ON, the ERROR indicator will be lit and the ASCII Unit will wait for the CTS and DSR signals to be turned ON. If the CTS signal is OFF during data transmission, the output operation of the ASCII Unit will be interrupted.

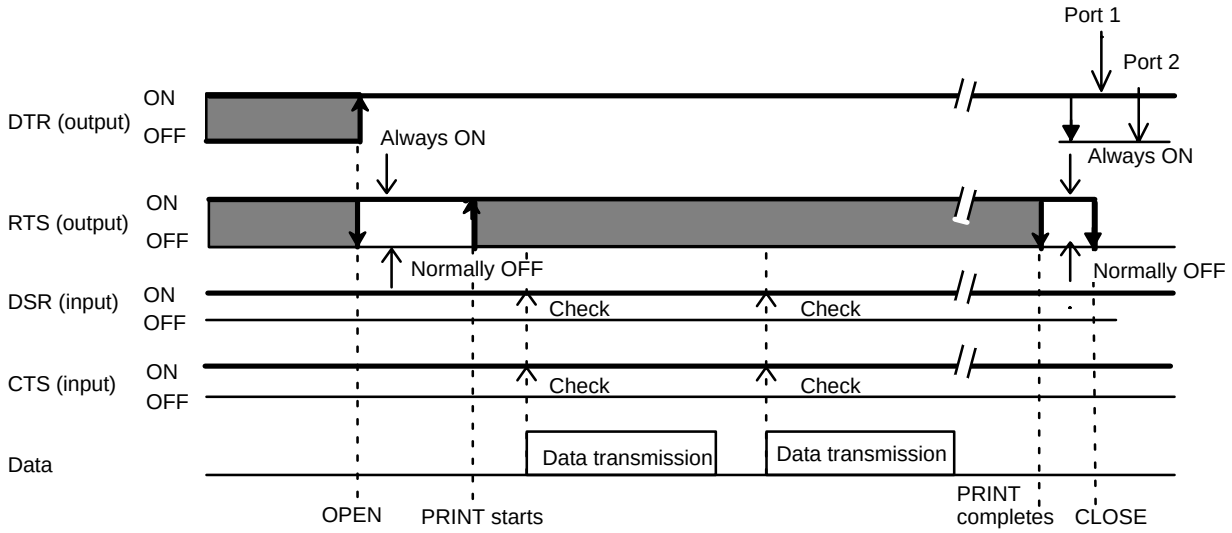
Note If the DSR or CTS signal is disabled, these signals will be ignored. However, if the CTS signal to Port 2 needs to be disabled, either turn it ON or connect Port 2 to the RTS signal.

If the CLOSE statement is executed, TERM is assigned to Port 1 and LPRT is assigned to Port 2.

The following timing chart applies if the peripheral devices are TERM and COMU when the OPEN instruction is executed.



The following timing chart applies when the peripheral devices are SCRN and LPRT when the OPEN instruction is executed.



Transmission from a Peripheral Device to the ASCII Unit

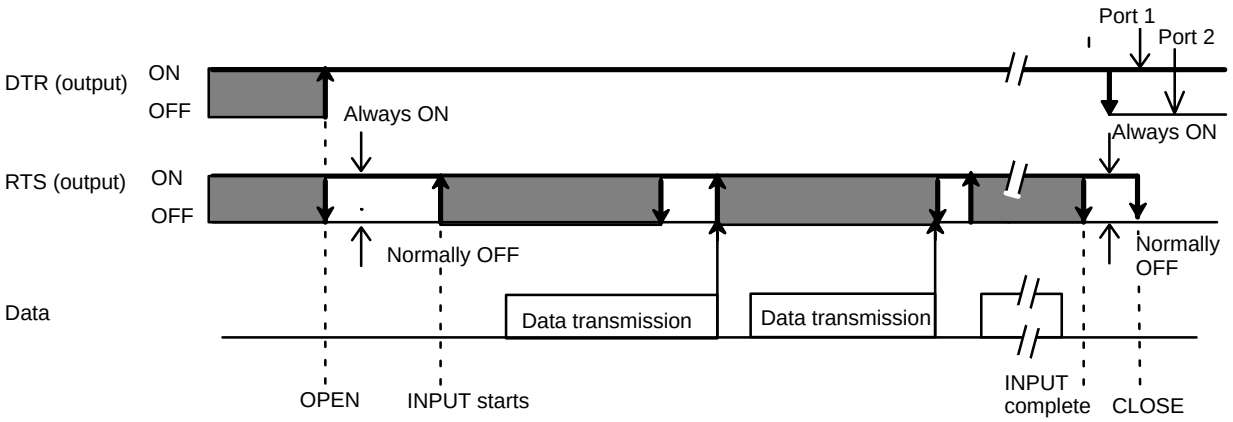
The DTR signal is ON when the KEYB or COMU is selected as the peripheral device with the OPEN instruction.

The RTS signal will be OFF if the effective signal wire is X (normally OFF) when the OPEN instruction is executed. If the effective signal wire is O (normally ON), the RTS signal will be ON from the execution of the OPEN instruction until the execution of the CLOSE instruction.

When the RTS signal is always ON, reception data will be stored in the buffer regardless of whether or not the INPUT instruction has been executed.

The INPUT, INPUT#, or INPUT$ instruction turns the RTS signal ON and data, if any, will be input. The CTS and DSR signals will not be checked.

When the CLOSE instruction is executed, Port 1 will be assigned to TERM and Port 2 will be assigned to LPRT.



Transmission from peripheral devices is possible when the RTS signal is ON.

Difference in Output According to Opened Peripheral Device

The following table shows the difference in instruction output, such as the PRINT instruction output, among the peripheral devices designated by the OPEN instruction. After RESET, Port 1 is assigned to TERM and Port 2 is assigned to LPRT automatically. There is no difference in output between Port 2 set to SCRN and Port 2 set to COMU.

○ Output
Δ Output with a code added
X Not output

Code		Abbreviation		TERM (see note 1)		SCRN (see note 2)		LPRT (see note 3)		COMU	
Hexa-deci-mal	Deci-mal			Port 1	Port 2	Port 1	Port 2	Port 1	Port 2	Port 1	Port 2
00	0		NUL	X	Not used	X	○	○	○	○	○
01	1	SH	SOM	X		X	○	○	○	○	○
02	2	SX	EOA	X		X	○	○	○	○	○
03	3	EX	EOM			X	○	○	○	○	○
04	4	ET	EOT	X		X	○	○	○	○	○
05	5	EQ	WRU	X		X	○	○	○	○	○
06	6	AK	RU	X		X	○	○	○	○	○
07	7	BL	BEL	X		X	○	○	○	○	○
08	8	BS	FEO	Δ (see note 4)		Δ (see note 4)	○	○	○	○	○
09	9	HT	TAB	X		X	○	○	○	○	○
0A	10	LF	LF	X		X	○	○	○	○	○
0B	11	HM	VT	○		○	○	○	○	○	○
0C	12	CL	FF	○		○	○	○	○	○	○
0D	13	CR	CR	Δ (see note 5)		Δ (see note 5)	○	○	○	○	○
0E	14	SO	SO	X		X	○	○	○	○	○
0F	15	SI	SI	X		X	○	○	○	○	○
10	16	DE	DCO	X	Not used	X	○	○	○	○	○
11	17	D1	XON	X		X	○	○	○	○	○
12	18	D2	TAP	X		X	○	○	○	○	○
13	19	D3	XOF	X		X	○	○	○	○	○
14	20	D4	TAP	X		X	○	○	○	○	○
15	21	NK	ERR	X		X	○	○	○	○	○
16	22	SN	SYN	X		X	X	○	○	○	○
17	23	EB	LEM	X		X	X	○	○	○	○
18	24	CN	CAN	X		X	○	○	○	○	○
19	25	EM	S1	X		X	○	○	○	○	○
1A	26	SB	EOF	X		X	○	○	○	○	○
1B	27	EC	ESC	X		X	○	○	○	○	○
1C	28	→	S4	○		○	○	○	○	○	○
1D	29	←	S5	○		○	○	○	○	○	○
1E	30	↑	S6	○		○	○	○	○	○	○
1F	31	↓	S7	○		○	○	○	○	○	○

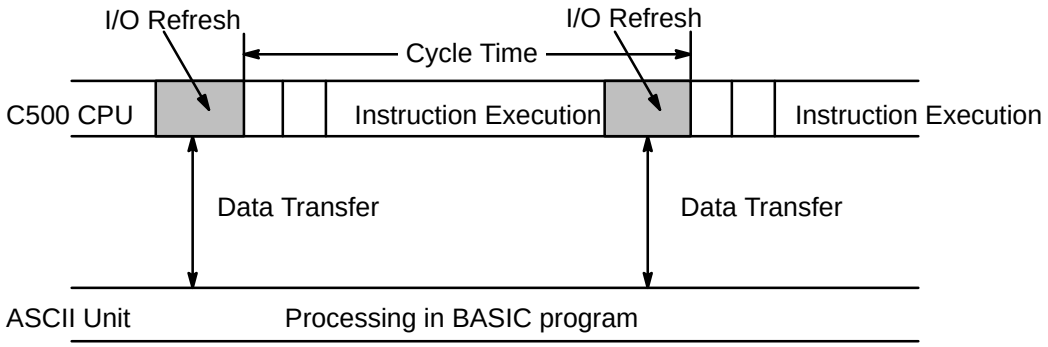
Note 1. Only port 1 can be assigned to TERM.
2. The SCRN outputs all codes from Port 2 except &H16 (cursor ON) and &H17 (cursor OFF).

3. If the LPRT receives the &H0A (LF), &H0B (HM), &H0C (CL), or &H0D (CR) code. &H0A (LF) will be added to the code and output. Any other code will be stored in the buffer and when the number of characters of the stored codes reaches 80, &H0A (LD) will be added to each of the codes and output. When the CLOSE instruction is executed, the port will close after all remaining data in the buffer is output.
4. A cursor shift code corresponding to the present display position is output and the cursor is shifted one character to the left.
5. The ASCII Unit outputs &H0C (CR) added with &H0A (LF).
6. If the COMU receives codes &H00 to &H1F, in the case of the C500-ASC01/02, the code is output immediately. Each code from &H20 to &HFF will be stored in the buffer and when the number of characters of the stored codes reaches 256, the codes will be output. In the case of the C500-ASC03/04, each code is output whenever the buffer receives the code.
7. If Port 1 is opened by a peripheral device other than TERM, be sure to execute the CLOSE instruction to stop the program. If key-in is not accepted from the terminal after the program stops, press the CTRL + X Keys.

Appendix C
PC Statements and Refresh Timing

Instructions and Refresh Timing

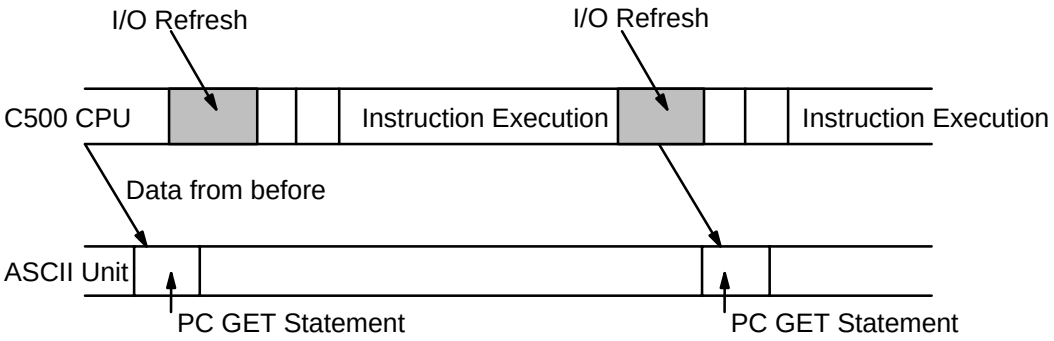
Data transfer between the ASCII Unit and the PC is executed during PC I/O refresh.



BASIC Statements and PC Cycle Time

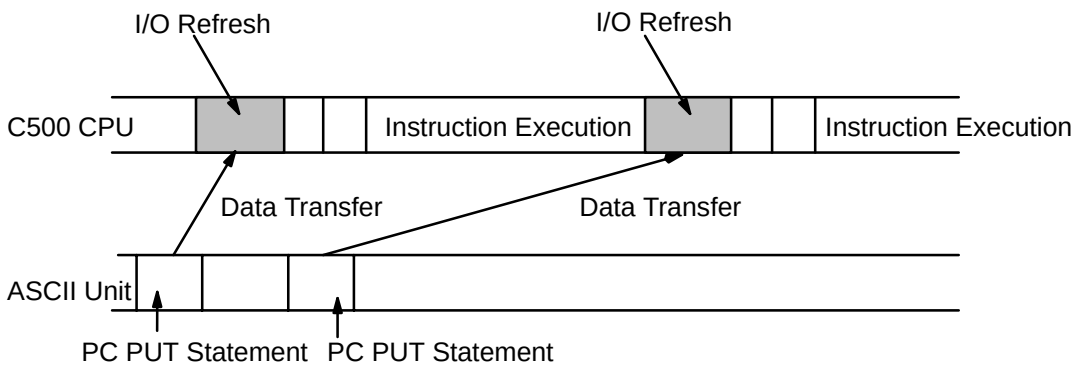
PC GET

The ASCII Unit takes in data obtained in the last PC I/O refresh before execution of PC GET.



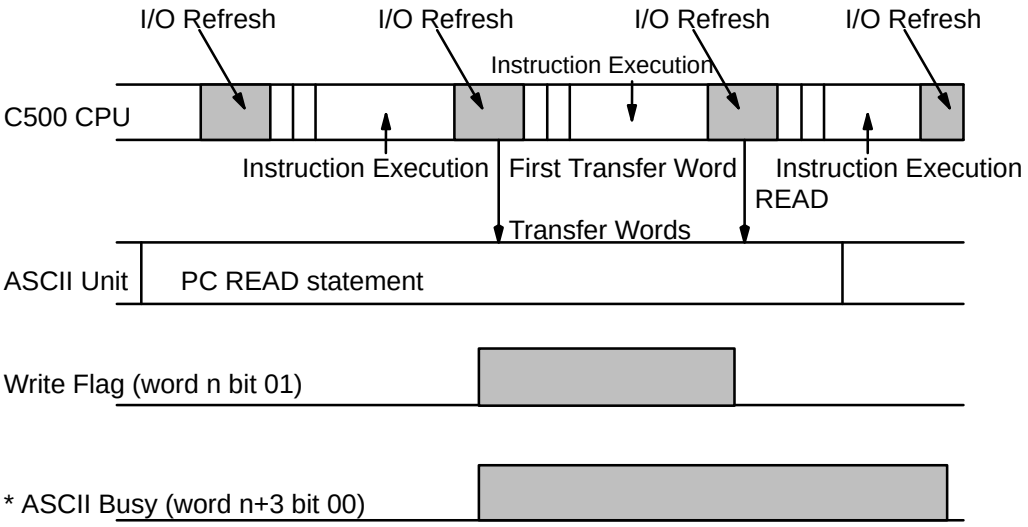
PC PUT

The ASCII Unit transfers data during the first PC I/O refresh after execution of PC PUT.



PC READ

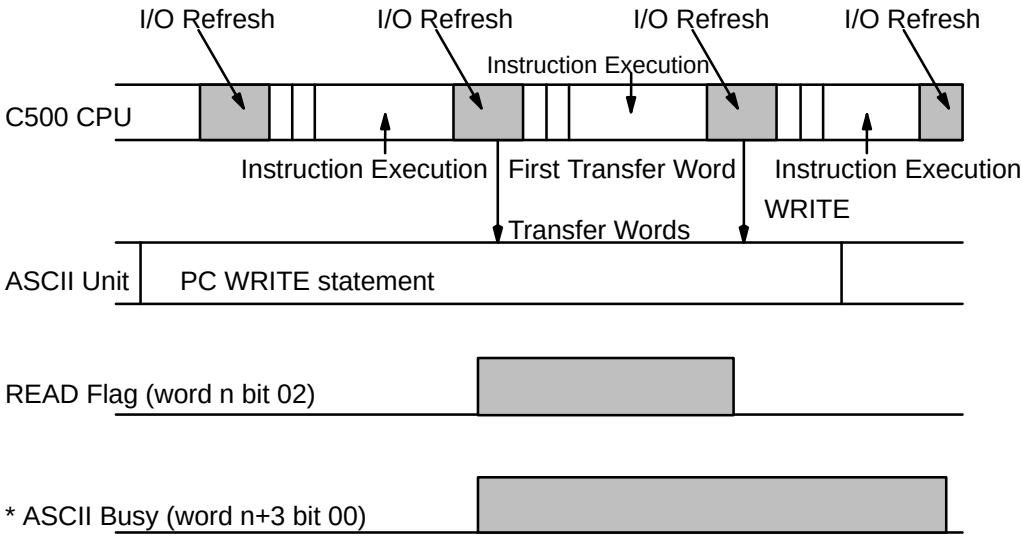
In four-word mode, when the PC's WRITE flag is set, the base address is transferred. By the next I/O refresh the data is read.



* When PC READ is executed in two-word mode using READ(88), n+3 becomes n+1.

PC WRITE

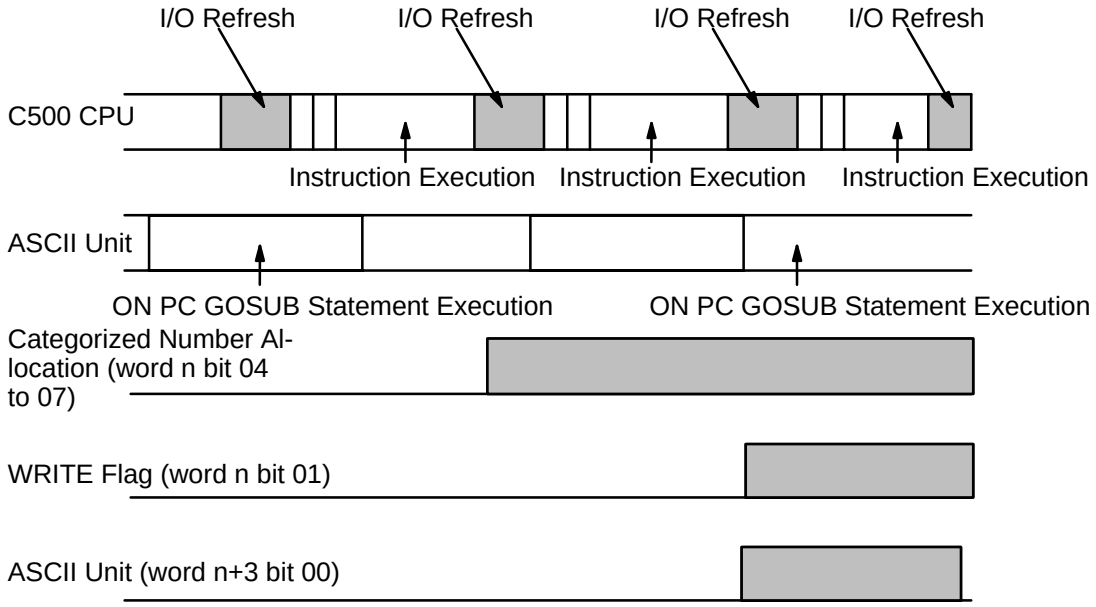
In four-word mode, when the PC's READ flag is set during I/O refresh, the PC WRITE statement obtains the base word address and the number of words to be transferred. With the next I/O refresh, data is transferred.



* When PC WRITE is executed in two-word mode using WRIT(87), n+3 becomes n+1.

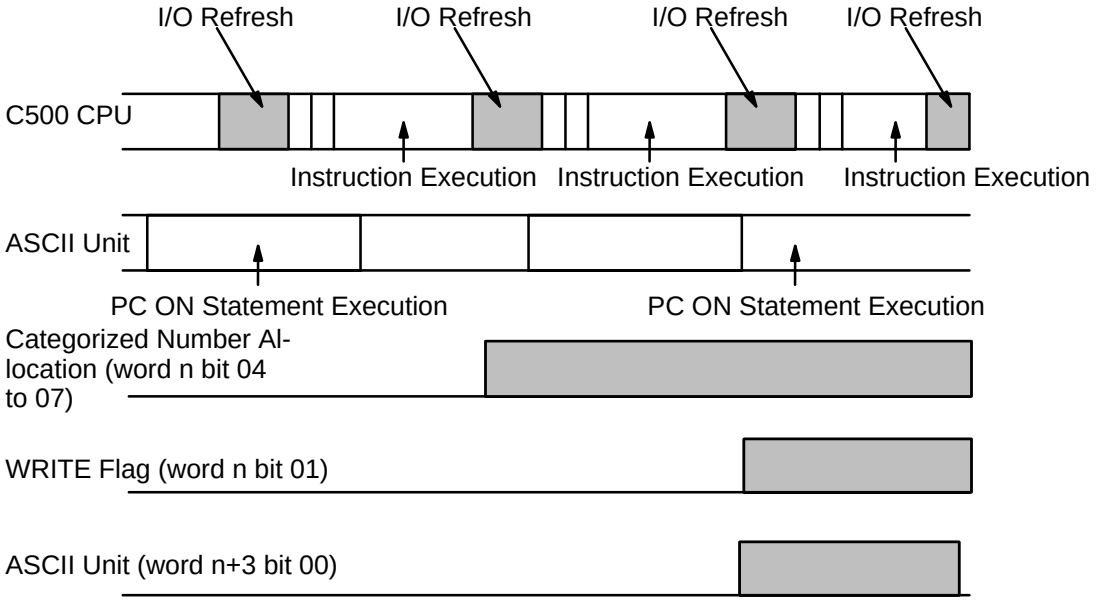
ON PC GOSUB

After the ON PC GOSUB statement is executed, the PC's categorized number allocation is written in. When the Write flag is set, the GOSUB statement is executed. Only when the WRITE flag is set will the ON PC GOSUB statement be executed.



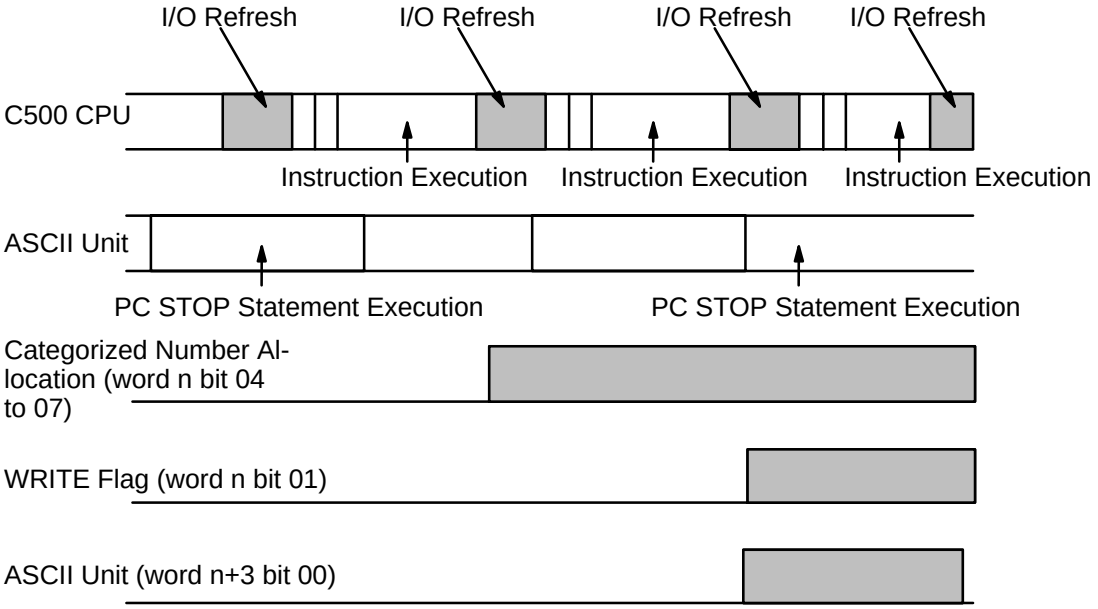
PC ON

After the ON PC GOSUB statement is executed, the PC's categorized number allocation is written in. When the Write flag is set, the GOSUB statement is executed. Only when the WRITE flag is set will the ON PC GOSUB statement be executed.



PC STOP

After the ON PC GOSUB statement is executed, the PC's categorized number allocation is written in. When the Write flag is set, the ASCII Unit busy flag is set for one cycle time, but the GOSUB statement is not executed. Only after the PC ON statement is executed will the ON PC GOSUB statement be executed.



Appendix D
Formatting and Data Conversion

Format	Meaning	Name
mIn	Indicates the nth byte of m decimal words	I format
mHn	Indicates the nth byte of m hexadecimal words	H format
mOn	Indicates the nth byte of m octal words	O format
mBn	Indicates the nth bit of of m binary words	B format
mAn	Indicates the nth byte of m ASCII words	A format
SmXn	Indicates the nth bit/byte of m words	S format

When m is omitted, the default value is one. When using the A format, one format designator corresponds to only one variable in the variable list: e.g., the first format designator corresponds to the first variable in the list, the second format designator corresponds to the second variable in the list, etc.

In all formats except A and S, one format designator can apply to many variables. For example:“5H2”; A, B, C, D, E. This is the same as “1H2, 1H2, 1H2, 1H2, 1H2”; A, B, C, D, E.

All format designators must be in uppercase characters.

Under normal conditions, the maximum number of words that can be transferred at one time is 255. When using the A or B formats, however, the maximum number of words that can be transferred is between 50 and 60.

I Format (mIn)

This format is used for decimal numbers (0 to 9):

m: number of words

I: decimal format designator

n: the nth digit of the word

Digit n	Bit															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	---				---				---				x 10^0			
2	---				---				x 10^1				x 10^0			
3	---				x 10^2				x 10^1				x 10^0			
4	x 10^3				x 10^2				x 10^1				x 10^0			

Example: 2I3 ... Indicates 2 decimal words of 3 digits each.

H Format (mHn)

This format is used for hexadecimal numbers (0 to F):

m: number of words

H: hexadecimal format designator

n: the nth digit of the word

Digit n	Bit															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	---				---				---				x 16^0			
2	---				---				x 16^1				x 16^0			
3	---				x 16^2				x 16^1				x 16^0			
4	x 16^3				x 16^2				x 16^1				x 16^0			

Example: 3H4 ... Three hexadecimal words of 4 digits each.

O Format (mOn)

This format is used for octal numbers (0 to 7):

m: number of words

O: octal format designator

n: the nth byte of the word

Digit n	Bit			
	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
1	---	---	---	x 8^0
2	---	---	x 8^1	x 8^0
3	---	x 8^2	x 8^1	x 8^0
4	x 8^3	x 8^2	x 8^1	x 8^0

Example: 4O2 ... Indicates four octal words of two digits each

B Format (mBn)

This format is used for binary numbers (0 to 1):

m: number of words

B: binary format designator

n: the nth bit of the word

Digit n	Bit															
	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	x2^0
1	—	—	—	—	—	—	—	—	—	—	—	—	—	—	x2^1	—
2	—	—	—	—	—	—	—	—	—	—	—	—	—	x2^2	—	—
3	—	—	—	—	—	—	—	—	—	—	—	—	x2^3	—	—	—
4	—	—	—	—	—	—	—	—	—	—	—	x2^4	—	—	—	—
5	—	—	—	—	—	—	—	—	—	—	x2^5	—	—	—	—	—
6	—	—	—	—	—	—	—	—	—	x2^6	—	—	—	—	—	—
7	—	—	—	—	—	—	—	—	x2^7	—	—	—	—	—	—	—
8	—	—	—	—	—	—	—	x2^8	—	—	—	—	—	—	—	—
9	—	—	—	—	—	—	x2^9	—	—	—	—	—	—	—	—	—
10	—	—	—	—	—	x2^{10}	—	—	—	—	—	—	—	—	—	—
11	—	—	—	—	x2^{11}	—	—	—	—	—	—	—	—	—	—	—
12	—	—	—	x2^{12}	—	—	—	—	—	—	—	—	—	—	—	—
13	—	—	x2^{13}	—	—	—	—	—	—	—	—	—	—	—	—	—
14	—	x2^{14}	—	—	—	—	—	—	—	—	—	—	—	—	—	—
15	x2^{15}	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Example: 5B14... Indicates five binary words of 14 bits each.

A Format (mAn)

This format is used for ASCII characters:

m: number of words

A: ASCII format designator

n: the nth byte of the word

Digit n	Bit	
	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
1	---	ASCII code
2	ASCII code	---
3	ASCII code	ASCII code

Example: 6A2... Indicates six ASCII words of two characters each.

A maximum of 255 words can be transferred at one time when the A format is used because many PC words can be represented by one BASIC variable.

Example: PC READ “50A3, 100A2, 30A1, 75A3”; A$, B$, C$, D$

A$: Fifty PC words (50 words x 2 characters = 100 characters) indicated by 50A3 are assigned to this variable.

B$: One hundred PC words (100 words x 1 character = 100 characters) indicated by 100A2 are assigned to this variable.

C$: Thirty PC words (30 words x 1 character = 30 characters) indicated by 30A1 are assigned to this variable.

D$: Seventy-five PC words (75 words x 2 characters = 150 characters) indicated by 75A3 are assigned to this variable.

S Format (SmIn, SmHn, SmOn, SmBn)

This format is used for array variables.

S: format designator

m: number of words

n: the nth bit/byte of the word

Format	Meaning
SmIn	Indicates an array in decimal format.
SmHn	Indicates an array in hexadecimal format.
SmOn	Indicates an array in octal format.
SmBn	Indicates an array in binary format.

Each S Format designator corresponds to one variable from the variable list: the first designator corresponds to the first variable in the list, etc.

The array variables must be one dimensional. Each array variable in the list must indicate (with a subscript) a specific element within the array. The number of words to be written to or read from the array will be incremented from the specified element. For example: if the array variable T(4) is specified in a READ statement and the corresponding format is S100I4, then 100 words will be read from the array, starting at T(4) and ending at T(104).

Example: PC READ “S100I4, S75H2, S80O3”; A(1), B(11), C(51)

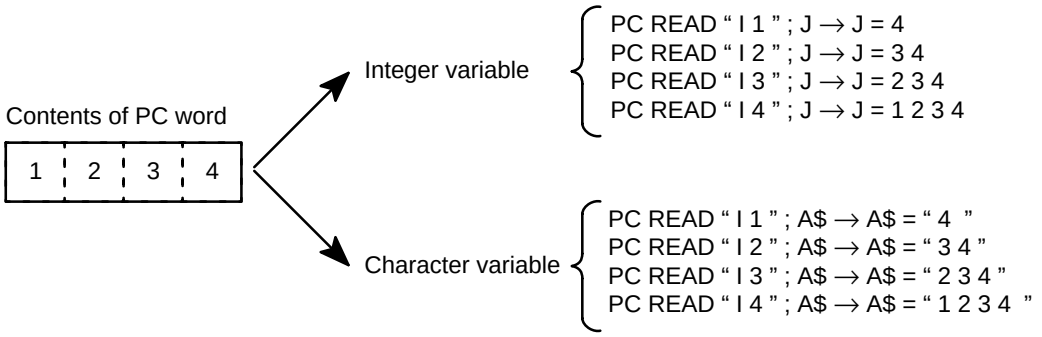
A(1) to A(100): A hundred words of 4-digit decimal data indicated by S100I4 are read to these variables.

B(11) to B(85): Seventy-five words of 2-digit hexadecimal data indicated by S75H2 are read to these variables.

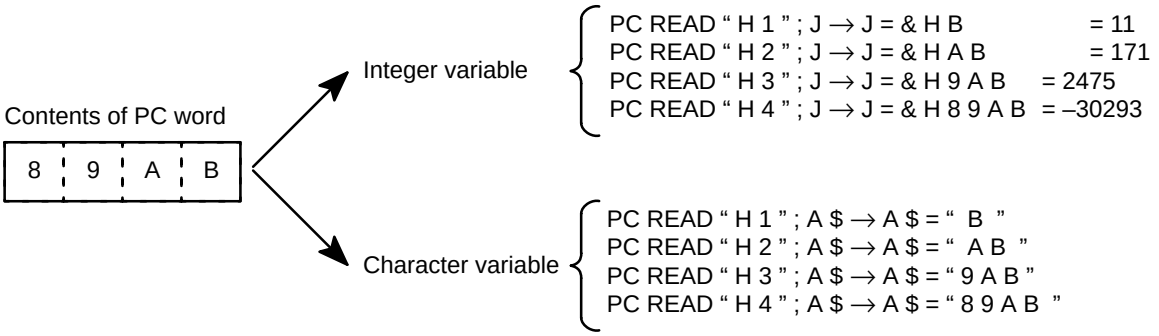
C(51) to C(130): Eighty words of 3-digit octal data indicated by S80O3 are read to these variables.

Examples of PC READ Format Conversion

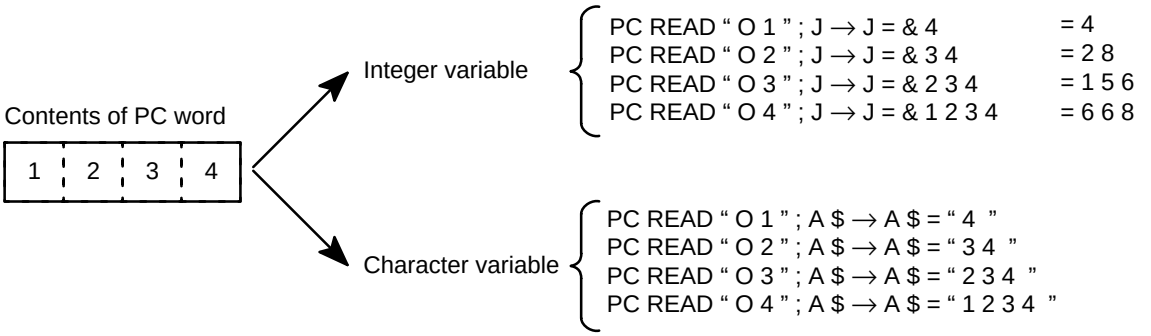
I Format



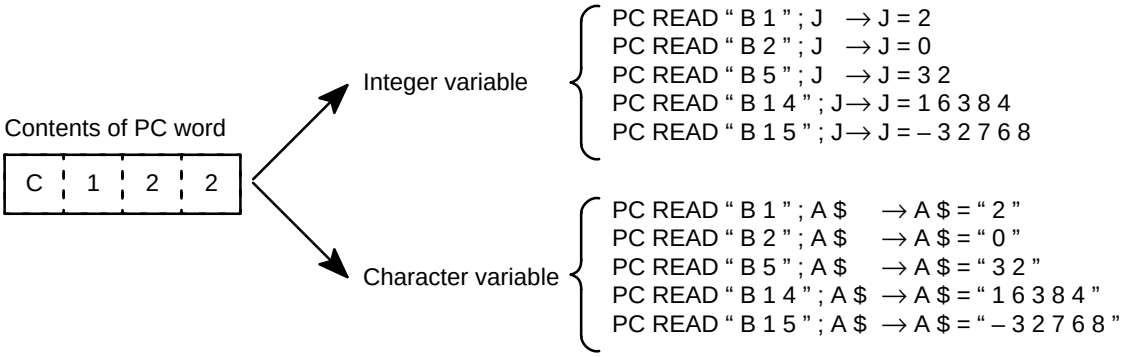
H Format



O Format

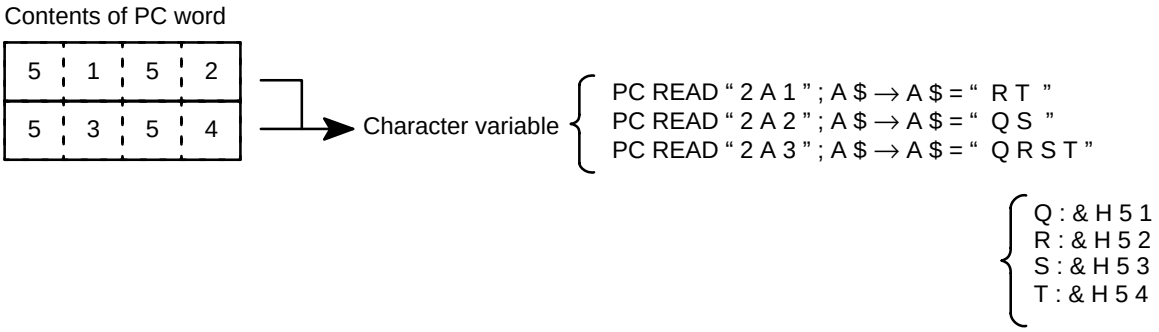


B Format

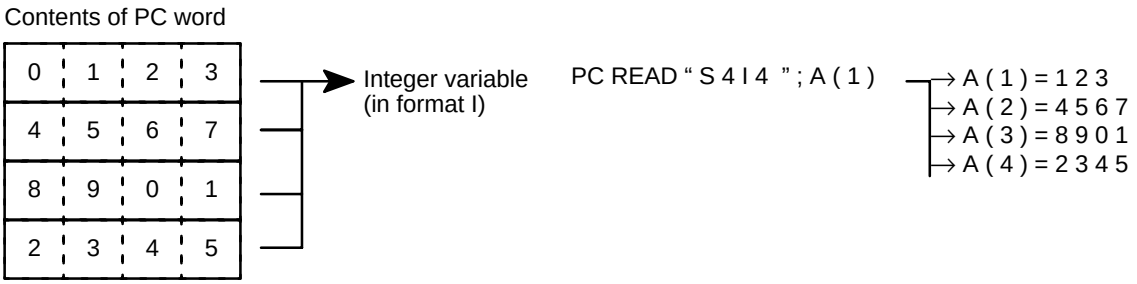


Note: The integer variable causes an error because it does not match the binary data format.

A Format

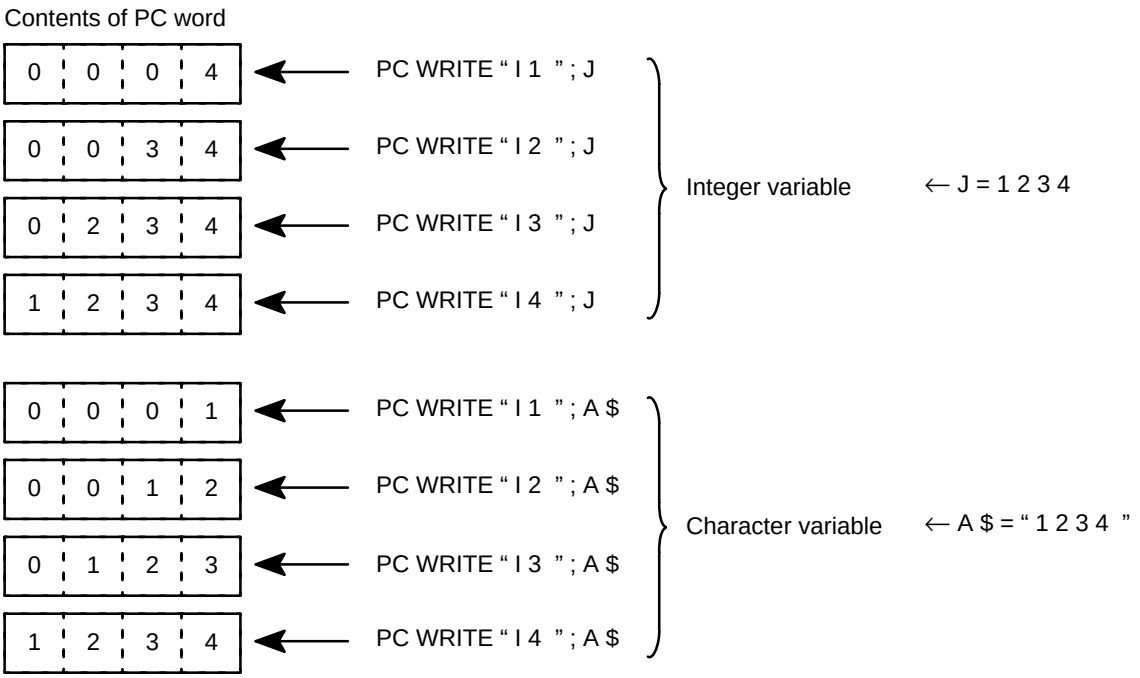


S Format

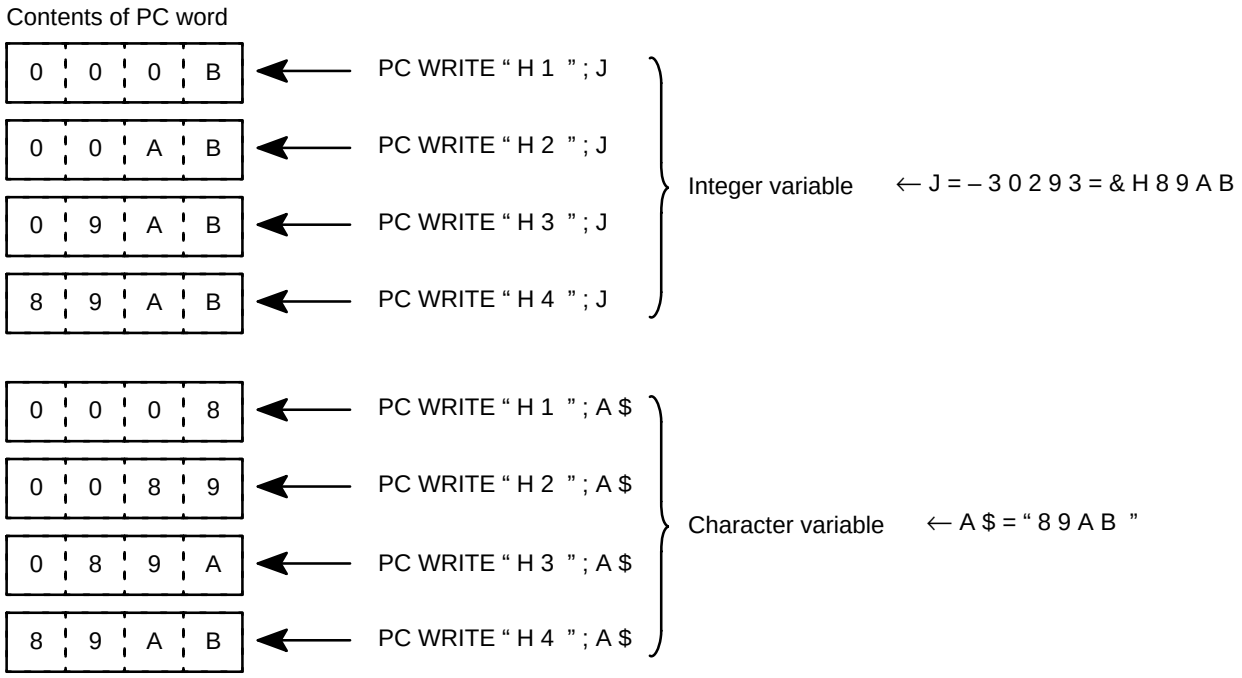


Examples of PC Write Format Conversion

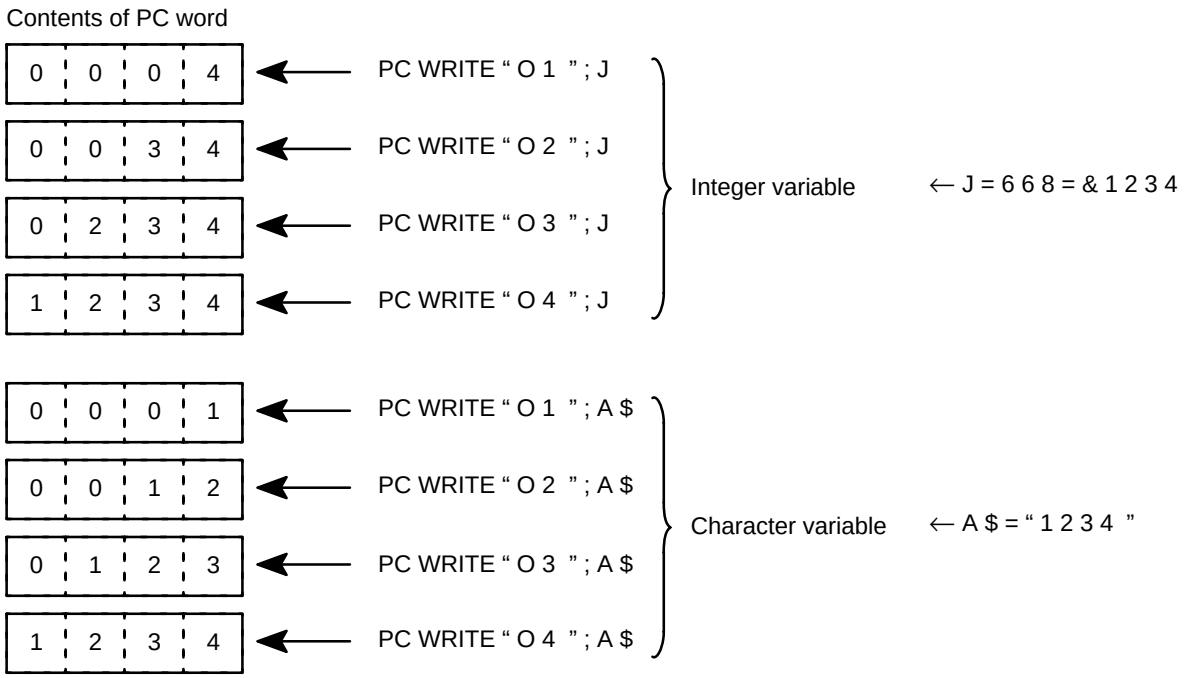
I Format



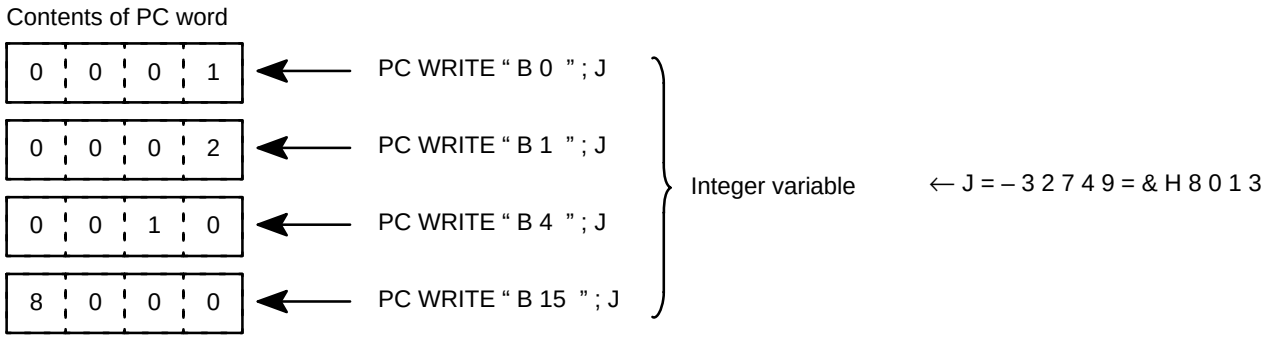
H Format



O Format

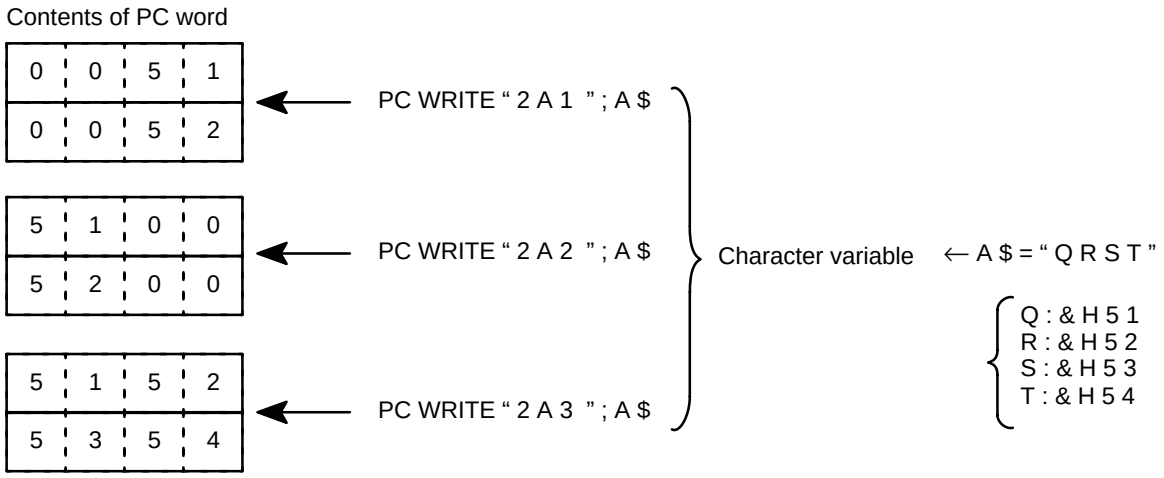


B Format

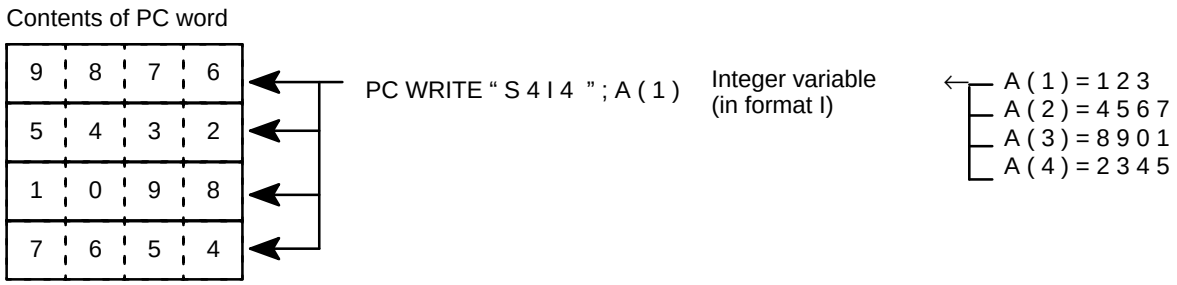


Note: Integer variables in B format will cause an error.

A Format



S Format



Execution Times

Command		Execution Time (ms)
PC READ	“ I 4 ” ; A	44 µs
PC READ	“ 5 I 4 ” ; A, B, C, D, E	0.2
PC READ	“ 1 0 I 4 ” ; A, B, C, D, E, G, H, I, J	0.4
PC READ	“ 1 0 0 A 3, 1 0 0 A 3, 5 5 A 3 ” ; A $, B $, C $	10.2
PC WRITE	“ I 4 ” ; A	42 µs
PC WRITE	“ 5 I 4 ” ; A, B, C, D, E	0.2
PC WRITE	“ 1 0 I 4 ” ; A, B, C, D, E, G, H, I, J	0.4
PC WRITE	“ 1 0 0 A 3, 1 0 0 A 3, 5 5 A 3 ” ; A $, B $, C $	10.4

The following table lists execution times for several different data transfer configurations using WRIT(87/191) and READ(88/190).

Instruction		1 word WRIT(87/191) #0001 00 DM 000	5 words WRIT(87/191) #0255 00 DM 000	10 words READ(88/190) #0001 DM 000 01	100 words READ(88/190) #0255 DM 000 01
C500	Executed	0.37 ms	3.64 ms	0.38 ms	3.66 ms
	Not executed	22 µs			
C1000H	Executed	1.3 ms	6.4 ms	1.3 ms	7.0 ms
	Not executed	6 µs			
C2000H	Executed	0.83 ms	4.21 ms	0.83 ms	4.62 ms
	Not executed	4 µs			
CV500	Executed	0.66 ms	4.91 ms	0.56 ms	4.63 ms
	Not executed	1.35 µs			
CV1000	Executed	0.55 ms	4.09 ms	0.47 ms	3.86 ms
	Not executed	1.13 µs			

Appendix E
Memory Map

This appendix provides the memory map of the ASCII Unit.

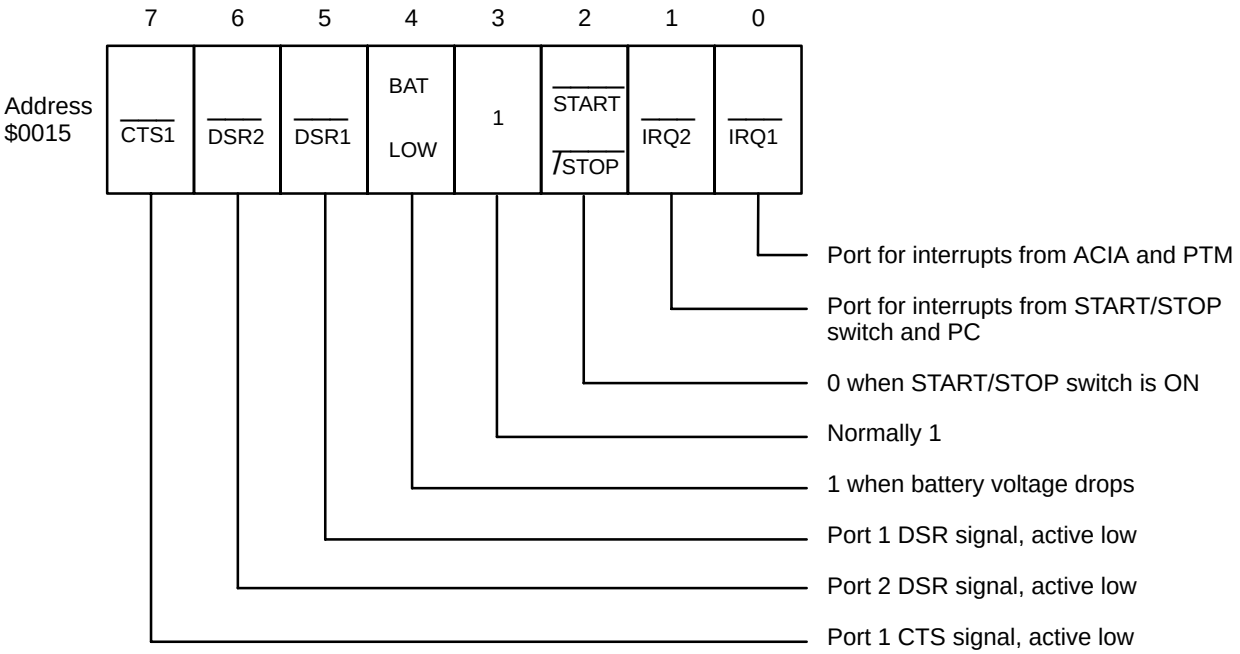
Memory Area	Base Address	Remarks
I/O area 1	&H0000	This area is for internal ports of the microprocessor 63B03.
System work area	&H0020	This area is used by the system.
Assembly language program area	&H2000	Stores assembly language program. The size of this area can be changed with MSET command.
BASIC Text area	---	Stores intermediate language codes of BASIC program. The size of this area can be changed with the MSET command.
System stack area	---	Stack area used by the system.
Character string area	---	Stores character strings. The size of this area is normally 200 bytes, but can be changed with the CLEAR command.
Common memory area or the Data Section	&H8000	RAM area for interfacing between ASCII Unit and PC. When this area is accessed, an I/O UNIT ERROR may occur. Do not access this area.
I/O area 2	&H9000	Area to which ports ACIA, PTM, and RTC are assigned.
System area	&HA000	This is the ROM area.

Port Address Assignments

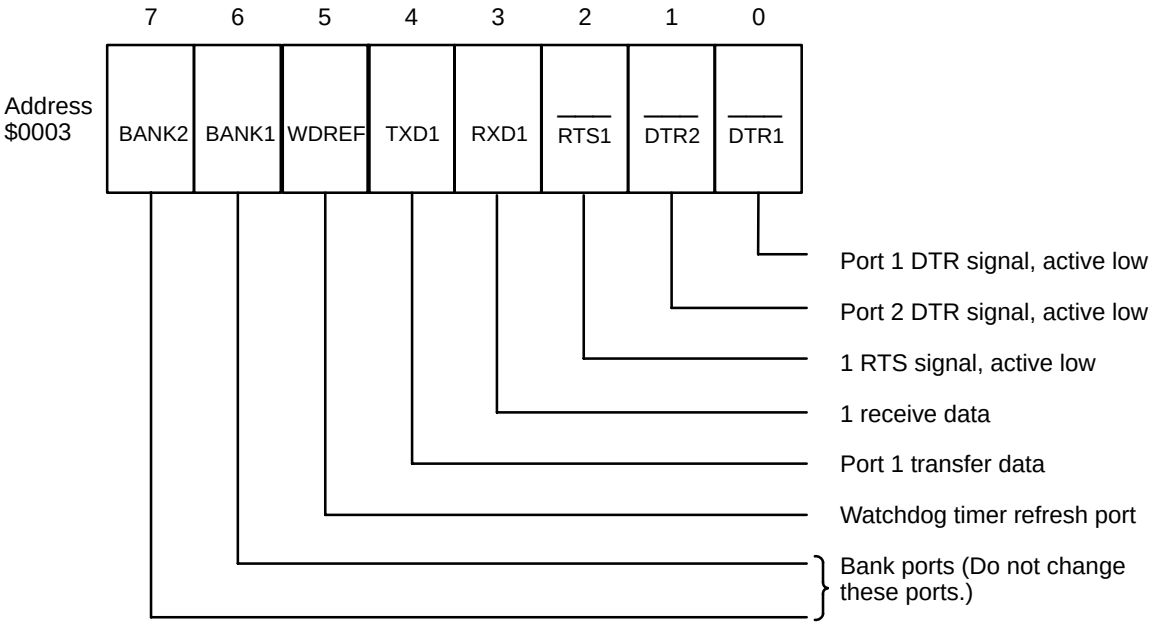
Address	R/W	Contents	System Default Value
$0010	R/W	Transfer rate/mode control register	$34
$0011	R/W	TX/RX control status register	$001
$0012	R	Receive data register	None
$0013	W	Transmit data register	None
$9400	R	Status register	None
$9400	W	Control register	$11
$9401	R	Receive data register	None
$9401	W	Transmit data register	None

Communication Flags

Communication Input Flags



Communication Output Flags



Devices

Address	R/W	Contents	System Default Value	Remarks
$9800	R	0	None	
	W	Control registers #1 and #3	$82	Writes to #3
$9801	R	Status register	None	
	W	Control register #2	$00	
$9802	R	Higher byte of timer #1 counter	None	
	W	Higher byte (MSB) of buffer register	None	
$9803	R	Lower byte (LSB) of buffer register	None	
	W	Lower byte of timer 1 latch	None	
$9804	R	Higher byte of timer #2 counter	None	
	W	Higher byte (MSB) of buffer register	None	
$9805	R	Lower byte (LSB) of buffer register	None	
	W	Lower byte of timer #2 latch	None	
$9806	R	Higher byte of timer #3 counter	None	
	W	Higher byte (MSB) of buffer register	None	Changes depend on transfer rate
$9807	R	Lower byte (LSB) of buffer register	None	
	W	Lower byte of timer #3 latch	None	

Address	R/W	Contents	System Default Value
$9000	R/W	1-second digit : 0 through 9	None
$9001	R/W	10-second digit : 0 through 5	None
$9002	R/W	1-minute digit : 0 through 9	None
$9003	R/W	10-minute digit : 0 through 5	None
$9004	R/W	1-hour digit : 0 through 9	None
$9005	R/W	10-hour digit : 0 through 2	None
$9006	R/W	1-day digit : 0 through 9	None
$9007	R/W	10-day digit : 0 through 3	None
$9008	R/W	1-month digit : 0 through 9	None
$9009	R/W	10-month digit : 0 and 1	None
$900A	R/W	1-year digit : 0 through 9	None
$900B	R/W	10-year digit : 0 through 9	None
$900C	R/W	Week digit : 0 through 6	None
$900D	R/W	Control register D	0 is set in D0.
$900E	R/W	Control register E	None
$900F	R/W	Control register F	0 is set in D0, 1, and 3.

Note A 4.9152-MHz clock is supplied to the MPU and a 1.2288-MHz clock is supplied to the ACIA and PTM.

Address	Contents	
$0145	Port 1	Port storage pointer (reception)
$0146		Data extraction pointer (reception)
$0147		Data storage pointer (transfer)
$0148		Reception buffer, 256 bytes
$024B	Port 2	Data storage pointer (reception)
$024C		Data extraction pointer (reception)
$024D		Data storage pointer (transfer)
$024E		Reception buffer, 256 bytes
$1440	Port 1	Transfer buffer, 256 bytes
$1540	Port 2	Transfer buffer, 256 bytes

Appendix F
Troubleshooting

Error Message Format

When an error occurs during BASIC program execution, the error messages shown in the following tables are output to the screen of the terminal. If a device other than a terminal is connected to port 1, the program stops, and the messages are reserved until the terminal is attached and CTRL+X is keyed in.

Example of a displayed message:

SYNTAX ERROR IN xxxx

xxxx is displayed when a command is executed with a number specified.

Error Message	Error Code	Explanation
BAD DATA IN PORT ERROR	58	Format of data read from port is wrong.
BAD I/O MODE ERROR	51	Wrong port or peripheral device has been specified.
BAD PORT DESCRIPTOR ERROR	55	Descriptor is incorrect.
BAD PORT NUMBER ERROR	50	Port number is incorrect.
BAD SUBSCRIPT ERROR	9	Subscript outside predetermined range is used. Assign subscript of maximum value with the DIM command.
CAN'T CONTINUE ERROR	17	Program execution cannot be resumed. Execute program with RUN command.
DEVICE I/O ERROR	53	Error has occurred during communication with a peripheral device.
DEVICE UNAVAILABLE ERROR	60	Wrong device name has been specified.
DIVISION BY ZERO ERROR	11	Attempt is made to divide data by 0.
DIRECT STATEMENT IN PORT ERROR	56	Unnumbered line has been read while program is being loaded.
DUPLICATE DEFINITION ERROR	10	Array, or user function, is defined in duplicate.
FORMAT ERROR	67	Incorrect format or memory area designator, number of words to be transferred or base address has not been specified.
FOR WITHOUT NEXT ERROR	23	FOR and NEXT statements are not correctly used in pairs.
ILLEGAL DIRECT ERROR	12	Attempt is made to execute statements that cannot be executed in direct mode. INPUT and LINE INPUT can be executed in BASIC program only.
ILLEGAL FUNCTION CALL ERROR	5	Statement or function is called incorrectly.
INPUT PAST END ERROR	54	All data in port has been read.
MISSING OPERAND ERROR	22	Necessary parameter is missing.
NEXT WITHOUT FOR ERROR	1	NEXT and FOR statements are not used in pairs.
NO RESUME ERROR	19	RESUME statement is missing in error processing routine.
NO SUPPORT ERROR	64	That operation is not supported.
OUT OF DATA ERROR	4	No data exists to be read by READ statement. Check number of variables in READ statements and number of constants in DATA statements.
OUT OF MEMORY ERROR	7	Memory capacity is full. Expand BASIC program area by CLEAR and MSET commands.
OUT OF STRING SPACE ERROR	14	Character area is insufficient. Expand area by CLEAR command.
OVERFLOW ERROR	6	Numeric value exceeds predetermined range.
PORT ALREADY OPEN ERROR	52	Port with specified number has already been opened. Attempt is made to open port more than once with the OPEN statement. Delete unnecessary OPEN statements.
PORT NOT OPEN ERROR	57	Unopened port or I/O device is specified. Open port with the OPEN statement.

Error Message	Error Code	Explanation
PROM ERROR	65	EEPROM is malfunctioning, or nothing is written in the EEPROM.
PROTECTED PROGRAM ERROR	62	Program is protected. To change program, delete name with PNAME command.
RESUME WITHOUT ERROR	20	RESUME statement is executed when no error exists.
RETURN WITHOUT GOSUB ERROR	3	RETURN statement is encountered before execution of GOSUB statement.
STRING FORMULA TOO COMPLEX ERROR	16	Character expression is too complex.
STRING TOO LONG ERROR	15	Character string is too long.
SYNTAX ERROR	2	Program does not conform to syntax.
TYPE MISMATCH ERROR	13	Variable types do not match.
UNDEFINED LINE NUMBER ERROR	8	Specified line number is wrong.
UNDEFINED USER FUNCTION ERROR	18	User function is not defined. Define execution start address with the DEF USR statement.
VERIFY ERROR	66	Error occurs during EEPROM verification.

Item	Cause	Correction
All Indicators do not light	Power to PC is OFF.	Turn ON power to PC.
	ASCII Unit is not mounted on PC securely.	Tighten mounting screws.
ERROR indicator is ON.	Power to peripheral device is OFF.	Turn ON power to device.
	Cable for device is disconnected.	Correctly connect cable, and tighten screws.
	Breakage in cable or faulty contact exists.	Repair or replace cable.
	Transfer rates and communication conditions of ASCII Unit and peripheral device do not match.	Correct transfer rates and communication conditions.
BAT ERR indicator is ON	Battery connector is disconnected.	Correctly connect battery connector.
	Battery voltage has dropped.	Replace battery.
Initial screen is <<PROGRAM MEMORY ERROR>>, and CTRL+X is ineffective.	BASIC program is damaged.	Press CTRL+I, and BASIC program will be erased. (If program is backed up in EEPROM, program can later be restored by LOAD command.)
Cannot program correctly	Operating System is damaged	Execute the following steps and then press the Reset switch. MSET &H2000 MON↵ 13A: 0_0↵ After executing these steps, turn off pin #2 on the front-panel DIP switch. The following message will be displayed on the initial screen: <PROGRAM MEMORY ERROR> When this message is displayed enter CTRL+I

Inspection Items

The following items should be periodically inspected.

Item	Particulars	Criteria	Remarks
Environment	Is ambient temperature appropriate?	0° to 55°C	Thermometer
	Is ambient humidity appropriate?	35% to 85% (without condensation)	Hygrometer
	Is dust built up?	Must be free from dust.	Visual inspection
Mounting condition	Are cable screws loose?	Must not be loose.	Standard screwdriver
	Is cable broken?	Must be mounted properly.	Visual inspection

Maintenance Parts

The battery life is 5 years at 25^{o}C. If the battery is used at higher temperatures, its life is shortened. When the battery voltage drops, the BAT ERR LED indicator blinks, and the Battery Low Flag (bit 06 of word n+1 in 2 word mode and n+3 in 4 word mode, where n = 100 + 10 x machine number) turns ON. Replace the battery within 1 week after the indicator blinks.

To replace the battery, take the following steps:

1. Turn OFF the power to the ASCII Unit. If power is not supplied to the Unit, apply power to the Unit for at least one minute and then turn it OFF.
2. Press the upper side of the battery storage cover, and slide it down to remove.
3. Disconnect the battery and connector and replace them with new ones.
4. Replace the battery storage cover.

Notes on Handling

Turning off the power to the PC before replacing the ASCII Unit.

When returning a defective unit to OMRON, inform us of the abnormal symptoms in as much detail as possible.

Appendix G
BASIC Commands, Statements, and Functions

The following tables list the BASIC commands, statements, and functions alphabetically.

The characters in the Command, Statement, and, Function columns denote the following:

Gen: General statement
Dev: Device Control statement
Arith: Arithmetic Operation function
Char: Character String function
Spec: Special function
Comm: Command

Item	Description	Command	Statement	Function	Page
ABS	Returns the absolute value of a number			Arith	54
ACOS	Returns the arc cosine of a number			Arith	54
ASC	Returns the value of the first character in a character string.			Char	56
ASIN	Returns the arc sine of a number			Arith	54
ATN	Returns the arc tangent of a number			Arith	54
AUTO	Automatically generates line numbers	Comm			28
CDBL	Rounds off a numeric value to make an integer			Arith	54
CHR$	Returns the character corresponding to the ASCII code given by the argument			Char	56
CINT	Converts a numeric value into a double-precision real number			Arith	54
CLEAR	Initializes numeric and character variables		Gen		33
CLOSE	Closes a port		Dev		51
CLS	Clears the screen		Dev		51
COM ON/ OFF/STOP	Enables, disables, or stops an interrupt from a communication port		Gen		34
CONT	Resumes execution of a program that has been stopped	Comm			28
COS	Returns the cosine of a number			Arith	54
CSNG	Converts a numeric value into a single-precision real number			Arith	55
DATA	Defines numeric and character variables for subsequent READ statements		Gen		34
DATE$	Sets or assigns the date			Spec	59
DAY	Sets or assigns the day (in numbers)			Spec	59
DEF FN	Defines and names a user-generated function		Gen		35
DEF INT/SNG/DBL/ STR	Declares the variable type as integer, single-precision, double-precision or string		Gen		35
DEF USR	Specifies the start address of the assembly language subroutine called from memory by USR		Gen		36
DEL	Deletes a line or portion of a line in the program	Comm			28
DIM	Specifies the maximum values for array variables and assigns the area		Gen		36
EDIT	Edits one line of the program	Comm			29
END	Terminates the execution of a program and closes all files		Gen		36
EOF	Verifies that the port buffer of the specified port is empty			Spec	60

Item	Description	Command	Statement	Function	Page
ERL/ERR	Returns the error code and the line number where the error has occurred			Spec	60
ERROR	Simulates an error and allows error codes to be defined		Gen		37
FIX	Returns the integer part of a number			Arith	55
FOR...TO... STEP␣NEXT	Repeats a FOR to NEXT loop a specified number of times		Gen		37
FRE	Returns the range of available memory			Spec	60
GOSUB␣ RETURN	Calls and executes the subroutine and returns to the original program line with a “RETURN” statement		Gen		38
GOTO	Branches to a specified line number		Gen		38
HEX$	Returns a string representing the hexadecimal value of the decimal argument			Char	56
IF...THEN... ELSE...GOTO ELSE	Selects the statement to be executed or branch destination as the result of an expression		Gen		38
INKEY$	Returns a character read from the keyboard			Spec	60
INPUT	Reads key input and assigns it to the specified variable		Gen		39
INPUT$	Returns a character string read from the keyboard and assigns it to the specified variable			Spec	60
INSTR	Searches for the first occurrence of a character string and returns its position			Char	56
INT	Shortens an expression to a whole number			Arith	55
KEY ON/OFF/STOP	Controls initiation, cancellation, and halting of key input interrupt		Gen		39
LEFT$	Returns a character string of the specified number of characters, beginning at the left of the string			Char	57
LEN	Returns the total number of characters in a specified character string			Char	57
LET	Assigns the result of the expression to the variable		Gen		40
LINE INPUT	Reads one line of input from the keyboard and assigns it to a character string variable		Gen		40
LIST/LLIST	Displays or prints a program	Comm			29
LOAD	Loads the program from the EEPROM or from a port	Comm			29
LOC	Returns the number of characters in the input queue waiting to be read			Spec	61
LOG	Returns the natural logarithm			Arith	55
MID$	Returns the specified number of characters starting from the specified character position		Gen	Char	41
MON	Sets the terminal to monitor mode	Comm			30
MSET	Sets the address boundary for an assembly program	Comm			30
NEW	Clears the program and all currently defined variables	Comm			31
OCT$	Returns a string which represents the octal value of the decimal argument			Char	57
ON COM GOSUB	Defines the branch destination of a subroutine invoked by an interrupt from a communication port		Gen		41
ON ERROR GOTO	Causes branching to the specified line in the event of an error		Gen		42
ON GOSUB GOTO	Causes branching to the specified line when “expression” is “true”		Gen		42

Item	Description	Command	Statement	Function	Page
ON KEY GOTO ON KEY GOSUB	Causes branching to the specified line when the specified key is input		Gen		43, 43
ON PC GOSUB	Defines an interrupt number and its associated subroutine branch line number		Gen		44
OPEN	Opens a port		Dev		51
PC GET	Reads data from the PC output area and assigns it to the specified variable		Gen		45
PC ON/STOP	Enables or stops an interrupt invoked by the PC		Gen		45
PC PUT	Writes the value of a numeric expression to the PC input data area		Gen		46
PC READ	Reads data from the specified PC memory area, converts it to the specified format, and assigns it to the specified variables		Gen		46
PC WRITE	Converts data to the specified format and writes it to the specified PC memory area		Gen		47
PEEK	Reads the contents of a specified memory address			Spec	61
PGEN	Sets the program memory area to be used	Comm			31
PINF	Displays the program area currently being used	Comm			31
PNAME	Names, or deletes the name, of the program selected	Comm			32
POKE	Writes data to a specified memory address		Gen		47
PRINT/LPRINT	Displays or prints the value of an expression		Gen		47
PRINT USING LPRINT USING	Displays or prints a character string in the specified format		Gen		48
RANDOM	Reseeds the random number generator		Gen		48
READ	Reads values from a data statement and assigns them to variables		Gen		49
REM	Inserts a comment statement into the program		Gen		49
RENUM	Reassigns line numbers in the program	Comm			32
RESTORE	Specifies which DATA statement will be used by the next READ statement		Gen		49
RESUME	Specifies the line where execution will resume after error processing		Gen		50
RIGHT$	Returns the number of characters in a string starting from the right			Char	58
RND	Returns a random number between 0 and 1			Arith	55
RUN	Executes the program	Comm			32
SAVE	Saves the program to the EEPROM or to a device connected to a communication port	Comm			33
SGN	Returns the sign of an argument			Arith	55
SIN	Returns the sine of a number			Arith	56
SPACE$	Returns an empty string of the specified number of characters			Char	58
STOP	Stops program execution		Gen		50
STR$	Converts a numeric value into a character string			Char	58
STRING$	Returns a character string of the specified length			Char	58
TAB	Outputs spaces up to the specified column position			Char	58
TAN	Returns the tangent of a number			Arith	56
TIME$	Sets or gives the time			Spec	58
TRON/TROFF	Specifies or cancels a program trace	Comm			33
USR	Calls an assembly language function routine defined by a DEF USR statement			Spec	62

Item	Description	Command	Statement	Function	Page
VAL	Converts a character string into a numeric value			Char	59
VERIFY	Verifies the program and the EEPROM contents	Comm			33
VARPTR	Returns the memory address where the variable is stored			Spec	63
WAIT	Sets a delay before the next command is executed		Gen		50

• MID$ Function is located on page 57

List of Program Examples

Programs in Two-word Mode

Example No.	Description	Page
1	To write data from the PC using the WRIT(87) to the ASCII Unit using the PC READ statement.	80
2	To write data from the ASCII Unit using the PC WRITE statement to the PC using the READ(88).	80
3	To enter characters from the keyboard and write them to the PC using the PC WRITE statement and WRIT(87).	81
4	The PC uses interrupt number 3 to direct the ASCII Unit to read five words of data from the specified DM addresses.	81
5	To read and print PC data at specific times using the ASCII Unit PC READ statement and WRIT(87)	81
6	To accept input from the keyboard and write it to the PC using the PC WRITE statement and READ(88)	82
7	To display the state of PC bit 1000 on a display device connected to port 2	82
8	To retrieve and print several types of data from the PC using the PC GET statement and WRIT(87)	83
9	To use PC interrupts to direct execution of the ASCII Unit	84
10	To print PC data and the time of data transfer	85
11	To input data from a bar code reader using the PC WRITE statement	85
12	To transfer data from the PC to the ASCII Unit with the ASCII Unit maintaining control	86
13	To transfer data from the ASCII Unit to the PC with the ASCII Unit maintaining control	87
14	To transfer data from the PC to the ASCII Unit with the PC maintaining control	88
15	To transfer data from the ASCII Unit to the PC with the PC maintaining control	89
16	To process data with the ASCII Unit	89
17	To transfer data input through the ASCII Unit keyboard to the PC and then back to the ASCII Unit after computations have been performed by the PC	91
18	To initiate data transfer with the START switch using the WAIT statement	92
19	To direct processing using different interrupts	92

Programs in Four-word Mode

Example No.	Description	Page
1	To print data at fixed time intervals using the LPRINT statement	93
2	To direct execution of the ASCII Unit from the PC using the PC GET statement	93
3	To control execution of the PC from the ASCII Unit using the PC PUT statement	94
4	To print out production data every hour from DM000.	95
5	To accept input from the keyboard and write it to the PC using the PC WRITE statement	95
6	To read data from an input file through a communication port	95
7	To transfer multi-word data from the ASCII Unit to the PC in four-word mode by using the PC WRITE statement continuously	96
8	To transfer multi-word data from the PC to the ASCII Unit in four-word mode by using the PC READ statement continuously	98

Assembly Language Example

Glossary

accumulator register — The arithmetic hardware register of the microprocessor.

ASCII Unit program — The BASIC program that runs the ASCII Unit and communicates with the PC program.

Backplane — A rack of hardware slots sharing a common bus line to which the CPU and all of its I/O Units are connected.

base address — The first address of a block of memory or data. When a block of data is to be transferred with one of the I/O commands, the base address must be specified.

baud rate — The speed at which data is transferred during I/O operations. The baud rate for the two ports is set with the right-side DIP switch. The standard baud rates are 300, 1200, 2400, 4800, 9600, and 19,200.

binary — The number system that all computers are based on. A binary digit can have only two values, zero and one. The octal and hexadecimal number systems are based on binary digits.

bit — The smallest piece of information that can be represented on a computer. A bit has the value of either zero or one. A bit is one binary digit.

boot program — The BASIC program that is automatically loaded into the ASCII Unit RAM upon power up or reset.

byte — A group of eight bits that is regarded as one unit.

communications port — A connector through which external peripheral devices can communicate with a host computer or microprocessor. The ASCII Unit has two communications ports used to connect to a personal computer, printer, or other I/O devices.

data transfer routine — The PC requires a dedicated data transfer routine incorporated into its program in order to communicate with the ASCII Unit. A data transfer routine is not necessary when the memory area designator parameter is used with the PC READ and PC WRITE statements.

data word — PC data is organized into units called words. Each word contains 16 bits and has a unique address in the PC memory. When transferring a block of data between the PC and the ASCII Unit, it is necessary to specify the address of the first data word in the block as well as the number of data words to be transferred. Throughout this manual the terms word and data word are used interchangeably.

device control codes — Keyboard strokes entered with the control key depressed that send control messages to peripheral devices such as a terminal display or a printer. For example, control codes can be used to position the cursor on a display or to cause the printer to print a line of text as it is being typed.

DIP switches — There are two sets of DIP switches on the back panel of the ASCII Unit. Each DIP switch has eight pins which can be set to either zero or one. These DIP switches are used for setting hardware parameters such as the baud rate and the start up mode.

EPROM/EEPROM — Nonvolatile memory (retains data when power is disconnected) is used for permanent storage of up to three ASCII Unit programs. If the start mode is set to

automatic, the boot program will be loaded to the RAM from the EPROM upon power up or reset. Programs can be read from and written to the EPROM with the LOAD and SAVE commands, respectively.

execution sequence The order of operation in which the PC and ASCII Unit hardware execute their respective programs.

flag A hardware flag is a bit that is set or cleared by the machine to indicate a particular state or condition of the Unit to a peripheral device or to the program. Examples of PC hardware flags are the Read and Write flags. A software flag is set or cleared by the user to indicate to the hardware a particular choice or option. For instance, software flags are sometimes used for setting the direction of data transfer or the baud rate of a communication device.

hexadecimal Hexadecimal or hex is a numerical system based on the number 16. One hex digit can be represented by four binary digits in the range of zero to 15. The numbers 10 through 15 are represented by the letters A through F, respectively.

index register One of the microprocessor's hardware registers. It is used for assembly language programming.

interrupt number A code that is sent from the interrupting device to the microprocessor indicating which device is "calling." The interrupt number is especially important if there is more than one peripheral device connected to a microprocessor.

interrupt A signal sent to the microprocessor from a peripheral device that causes the microprocessor to alter its normal processing routine. An interrupt says to the microprocessor, "stop what you're doing and pay attention to me !" When an interrupt is acknowledged by the microprocessor, program execution will branch to an interrupt service routine specifically written to handle the given interrupt.

I/O device I/O stands for input/output. Some examples of I/O devices are printers, modems, fax machines, and display terminals.

machine no. switch Used to select the unit number for the allocation of PC words. The Machine No. switch is located on the front panel of the ASCII Unit.

mantissa The part of a numerical expression to the right of the decimal point.

memory area designator (@) A parameter of the PC READ and PC WRITE statements used to access specific PC data areas. When using the memory area designator for data transfer, the ASCII Unit does not need an accompanying PC data transfer routine.

monitor mode The mode or environment where assembly language programs are written, edited, and tested.

monitor mode commands The commands used in monitor mode for writing, editing, and debugging an assembly language program.

MSB/LSB MSB stands for Most Signicant Byte and refers to the upper or left half of a data word (a data word contains two bytes). The Least Significant Byte refers to the lower or right half of a data word.

octal A numerical system based on the number eight. One octal digit is made up of three binary digits in the range of zero to seven.

parameter/argument A parameter is a value or symbol supplied to a BASIC or assembly language command. A parameter either directs a command to implement a particular op-

	tion or format, or supplies a memory address where data can be stored. Similar to a parameter and sometimes used interchangeably is the term “argument”. Where a parameter usually supplies some type of control information to the function or command, an argument is usually a variable that supplies needed data.
PC program	A program that runs the PC; it is written in the Ladder Diagram programming language.
polling	A process whereby the microprocessor periodically checks the value of a specified bit or byte, and depending on that value, the microprocessor takes some specified action.
port buffer	Special memory that is used to temporarily store data that has just been received or is about to be sent out through a communication port.
program counter	A microprocessor register that keeps track of program execution. It is used for assembly language programming.
RAM	Stands for Random Access Memory and is used for running the ASCII Unit program. RAM will not retain data when power is disconnected. Therefore data should not be stored in RAM.
Read Flag	A PC hardware flag that indicates when data can be read from the PC. When this flag is set, data can be accessed by a peripheral device.
reading/writing	When something is read, it is taken or copied from a remote location and brought to the reference point. When something is written, it is sent from the reference point to a remote or peripheral device.
RS-232C interface	The industry standard connector for serial communications. The ASCII Unit communication ports use RS-232C connectors.
cycle time and refreshing	The PC is constantly scanning through its program, checking all of its inputs and adjusting its outputs. The time required for the PC to run through its program one time is called the cycle time. Each time the PC completes one cycle of its program, it updates or refreshes its outputs. The ASCII Unit cannot read data from the PC during data refresh.
stack pointer	A microprocessor index register used for assembly language programming.
start address	The starting address of a block of data. This term is used as a parameter in many of the assembly language monitor mode commands**.**
start mode	Indicates how the ASCII Unit starts up when power is first applied or the Unit is reset. The two choices are manual mode and automatic mode. The mode can be selected by setting pins one and two of the left-side DIP switch.
START/STOP switch	A toggle switch on the front panel of the ASCII Unit used for starting and stopping execution of the ASCII Unit program.
upload/download	Upload usually refers to the transfer of a program or information from a remote device to a computer or other controlling device. Download usually refers to data transfer from a computer or other controlling device to a remote device. From the users point of view, if data is being sent to another device, it is being downloaded. If data is being received from another device, it is being uploaded.
valid signal line	A parameter of the OPEN command which specifies which communication signals (CTS, DSR, RTS) are to be used for handshaking.

watchdog timer A clock on the PC that measures the time it takes the PC program to complete one cycle. If the cycle time is longer than 100 ms, a warning is issued. If the cycle time is longer than 130 ms, the PC will suspend operation. The watchdog timer is reset at the beginning of each cycle.

word A word is made up of two bytes or 16 bits. The term “word” is used interchangeably with the term “data word” to indicate a single unit of data. Blocks of data are transferred in word units. For data transfer, the address of a data block’s first word and the number of words to be transferred must be specified.

Write Flag A PC hardware flag that indicates when data can be written to the PC. When this flag is set, data can be written to the PC.

XON/XOFF OPEN statement parameters that control the rate at which the port buffers receive and transmit data. If the XON command is specified to be ON by the OPEN statement, then when the port buffer becomes 3/4 full, the ASCII Unit will suspend data transfer until the port buffer is less than 1/4 full. In a case where a transmitting device is sending data at a faster baud rate than the ASCII Unit is set for, the XON command will keep the transmitted data from being written over.

Index