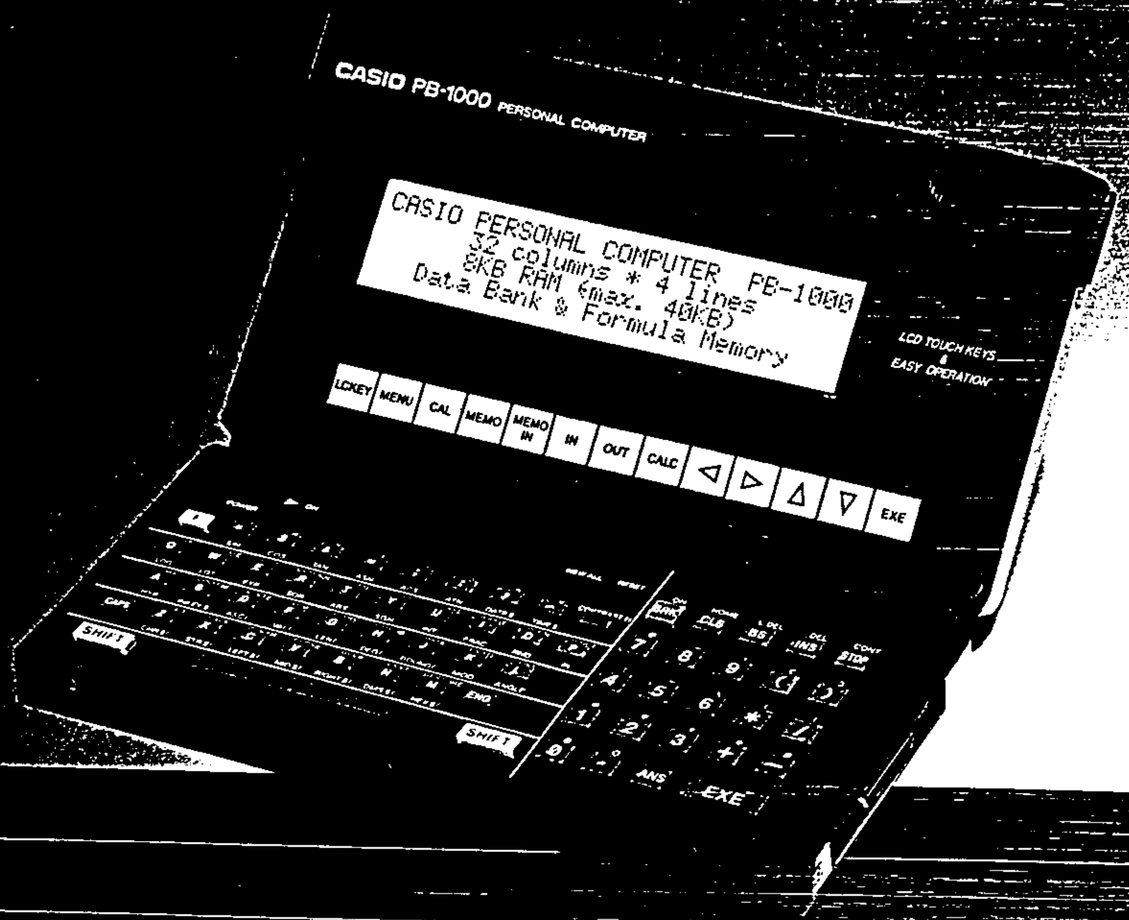
MASTER

CASIO

COMMAND REFERENCE


CASIO PB-1000 PERSONAL COMPUTER
CASIO PERSONAL COMPUTER PB-1000
32 columns * 4 lines
8KB RAM (max. 40KB)
Data Bank & Formula Memory
LCD TOUCH KEYS & EASY OPERATION
LCKEY
MENU
CAL
MEMO
MEMO IN
IN
OUT
CALC
EXE
SHIFT
CLS
BS
INS
STOP
ANS

PERSONAL COMPUTER

PB-1000

英

PERSONAL COMPUTER

PB-1000

COMMAND REFERENCE

CASIO

Introduction

This manual describes the syntax of C61-BASIC and HD61700 Assembler, the programming languages for the PB-1000.

The PB-1000 Owner's Manual is provided together with this manual. Please first read the owner's manual to familiarize yourself with fundamental handling, programming using C61-BASIC, and assembler. This manual will serve as a handy reference for those who are familiar with BASIC and/or assembler programming.

CONTENTS

PART 1 C61-BASIC 1
1-1 FEATURES OF C61-BASIC 2
1-2 CONFIGURATION OF BASIC PROGRAMS 3
Operating Mode 3
Direct Mode 3
Program Mode 3
Line Numbers 3
Statements 3
1-3 CONSTANTS 4
Numeric Constants 4
Character Constants 4
1-4 VARIABLES 4
Variable Names and Array Names 4
Arrays 5
Handling of Variables and Arrays 5
1-5 OPERATORS 5
Arithmetic Operators 6
Relational Operators 6
Logical Operators 6
String Operators 7
Order of Precedence 7
1-6 TOUCH-KEYS 8
1-7 PLOTTER COMMANDS 9
DIP Switches 9
Printer Mode 10
Plotter Mode 11
Example of Printing Using the Plotter Mode 13
1-8 COMMUNICATIONS 14
RS-232C Interface 14
File Descriptor 14
Data Input/Output 16
List of BASIC Commands 16
1-9 DATA FILES 17
Sequential Files 17
Random Files 17
BASIC COMMANDS 19
MANUAL COMMANDS 20
PASS 20
NEW 21
SYSTEM 21
CLEAR 22
MON 24
DELETE 24

LIST 25
EDIT 26
VARLIST 26
RUN 27
TRON 28
TROFF 28
FUNDAMENTAL COMMANDS 29
END 29
STOP 30
GOTO 31
GOSUB 32
RETURN 33
ON GOTO 34
ON GOSUB 35
IF~THEN~ELSE/IF~GOTO~ELSE 36
FOR~NEXT 37
REM(') 38
LET 39
DATA 40
READ 41
RESTORE 42
PRINT 43
TAB 44
REV 45
NORM 45
PRINT USING 46
LOCATE 47
CLS 48
DEFCHR$ 49
BEEP 50
INPUT 51
INKEY$ 52
INPUT$ 53
DIM 54
ERASE 55
CALL 56
PEEK 56
POKE 57
DRAW/DRAWC 58
POINT 59
ON ERROR GOTO 60
RESUME 61
ERL 62
ERR 62

NUMERIC FUNCTIONS 63
ANGLE 63
SIN, COS, TAN 64
ASN, ACS, ATN 65
HYP SIN, HYP COS, HYP TAN 66
HYP ASN, HYP ACS, HYP ATN 67
EXP 68
LGT, LOG 69
SQR 70
ABS 70
SGN 71
INT 72
FIX 73
FRAC 73
ROUND 74
PI 74
RND 75
CHARACTER FUNCTIONS 76
CHR$ 76
ASC 76
STR$ 77
VAL 78
MID$ 79
RIGHT$ 80
LEFT$ 81
LEN 82
HEX$ 82
&H 83
DEG 84
DMS$ 84
TIME$ 85
DATE$ 86
STATISTIC FUNCTIONS 87
STAT 87
STAT CLEAR 88
CNT 88
SUMX, SUMY, SUMX2, SUMXY 89
MEANX, MEANY 90
SDX, SDY, SDXN, SDYN 90
LRA, LRB 91
COR 91
EOX, EOY 92
I/O COMMANDS 93
LLIST 93
LPRINT 94
LPRINT USING 95
OPEN 96

1-1 FEATURES OF C₆₁-BASIC

C₆₁-BASIC is based on Japan Industrial Standard (JIS) BASIC (C6207) which has recently been officially formulated. This is a powerful version of the BASIC language, with enhanced arithmetic functions and filing capabilities.

- **C₆₁-BASIC shares the following features with ordinary BASIC.**

1. Ease of Understanding
The syntax of BASIC is easier for beginners to use.

2. Easy Program Writing
Writing and modifying programs can be performed easily with interaction between the programmer and computer which provides feedback to the user as the program is written. This allows the creation of simple and easy-to-understand programs.

- **In addition, the following features have been added to C₆₁-BASIC.**

1. High Precision Arithmetic
Numeric values are displayed on the screen with 10-digit mantissas and 2-digit exponents (internal calculation used 13-digit mantissa and 2-digit exponent).

2. Various scientific functions and paired-variable statistical functions

a) The following scientific functions may be used in arithmetic operations.

SIN COS TAN ASN ACS ATN HYPSIN HYPCOS HYPTAN
HYPASN HYPACS HYPATN LOG LGT EXP SQR ABS SGN
INT FIX FRAC PI ROUND RND

b) C₆₁-BASIC provides users with the following string functions.

CHR$ STR$ MID$ LEFT$ RIGHT$ HEX$ ASC VAL LEN

c) Statistical functions including linear regression.

3. BCD Arithmetic
BCD arithmetic functions are provided for unequaled calculation accuracy for business, scientific, and technical applications.

4. Memory File Function
Since multiple programs can be stored independently in main memory so programs do not have to be loaded into memory each time they are run.

5. Extended Variable Names
Variable names can consist of up to 255 characters, including upper case and lower case letters. Programs become easier to understand if variable names that indicate the contents of variables are used.

6. Enhanced Debugging Functions
A TRON command displays the current line number of the program during execution to aid in following program flow and spotting bugs.

7. Powerful Screen Editor
A powerful screen editor allows easy program modification by rewriting the program as it is displayed on the screen using keyboard input. This function takes full advantage of the interactive characteristics of the BASIC language.

8. Integrated Control of Peripheral Devices
The file management concept is adopted to facilitate easy control of peripheral devices such as floppy disk drives. All exchanges of data with peripheral devices are conducted through file transfers, so virtually the same concept can be applied throughout when using any peripheral device.

9. Touch-Key Functions for Improved Operability
Touch-keys on the LCD display facilitate programming by allowing the operator to select operations by simply pressing a key on the screen.

10.Timer Function
A built-in time makes it possible to execute programs that require timing.

1-2 CONFIGURATION OF BASIC PROGRAMS

Operating Mode

The message "Ready" appears on the display screen when BASIC is activated. In this mode, the computer is ready to receive commands. Since commands can be directly entered and executed when "Ready" is shown on the screen, this is called the command mode. In addition, the state when an integrated program is executed by entering successive line numbers is referred as the command mode.

Direct Mode

This mode executes BASIC without entering any line numbers. Execution begins after the command is input and the EXE key is pressed.

Program Mode

Whenever a line number is included with a command or commands, both the line numbers and command are stored together in memory in ascending order. A series of numbered commands stored in memory is called a program, and programs are executed using the RUN command.
Each command line can contain up to 255 characters, and a space must be included between the line number and command, so each command line has a maximum capacity of 254 characters of user-designated input.

Line Numbers

The range of possible line numbers is 1 to 65535.

Statements

A statement is the smallest executable unit in BASIC, and consists of commands, expressions and functions. Two or more statements can be joined together by colons to form what is known as multistatements.

1-3 CONSTANTS

Numeric Constants

- **Expressing Numeric Values**

a) Decimal notation

b) Hexadecimal notation

Numeric values written in the order of sign, &H, and a 1 ~ 4 digit hexadecimal expression (0 ~ 9, A ~ F).

- **Numeric Precision**

a) Internal Numeric Calculations

13-digit mantissa and 2-digit exponent. However, PI is handled as an 11-digit mantissa (i.e. 3.1415926536).

b) Calculation Results

Rounded off to a 10-digit mantissa and 2-digit exponent.

c) Character Input Capacity

Up to 255 characters per line.

d) Internal Rounding

Internal calculation uses a 13-digit mantissa. For numbers in ordinary arithmetic calculations, the 11th, 12th and 13th digits are rounded off when they are 049 or less, and rounded up when they are 950 or more.

e) Calculation Result Display

When the exponent of an integer is less than 1 × 10^{10}: Integer expression

For decimal numbers with 10 digits or less: Decimal expression

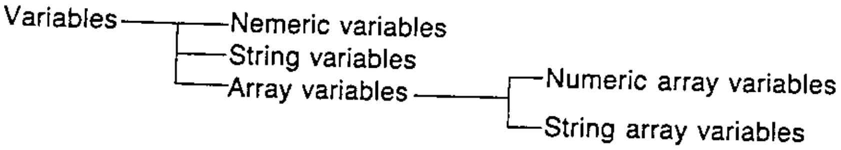
Other numbers: Exponential expression

Rounded expressions: The 11th digit is rounded off when the results of calculations are displayed.

String Constants

Strings enclosed in quotation marks.

1-4 VARIABLES



Variable Names and Array Names

Variable names and array names:

a) Are character strings with an upper case alphabetic character or lower case alphabetic character the leading (first) position.

b) Are composed of upper or lower case alphabetic characters or numbers following the leading alphabetic character.

c) Cannot use reserved words as the leading characters.

d) Can be up to 255 characters long.

Arrays

a) Arrays are declared by a DIM statement.
b) Subscripts in an array are positive integers, and fractions are truncated.
c) The dimensions of an array are only limited by the capacity of internal stacks.
d) The maximum value of subscripts is only limited by memory capacity.

Handling of Variables and Arrays

a) The same variables and/or arrays can be used different programs.
b) String variables are stored in the character data area specified by CLEAR statements.
c) Array variables may not be used unless they are first declared.
d) FIELD variables (see page 103) are only valid while a file is open, and they receive nulls when the file is closed. However, FIELD variables take on a character length of three when the file is closed.

1-5 OPERATORS

Operators	Arithmetic operators	Signs	+, −
		Addition	+
		Subtraction	−
		Multiplication	*
		Division	/
		Exponentiation	∧
		Integer division	¥
		Integer remainder of integer division	MOD
	Relational operators	Equal to	=
		Does not equal	< >, > <
		Less than	<
		Greater than	>
		Less than or equal to	= <, < =
		Greater than or equal to	= >, > =
	Logical operators	Negation	NOT
		Logical product	AND
		Logical sum	OR
		Exclusive OR	XOR
	String operator		+

Arithmetic Operators (+, −, ∗, /, ∧, ¥, MOD)

- Fractions are truncated in ¥ and MOD calculations, when the operands on both sides of the operator are not integers.
- In ¥ and MOD calculations, the division is performed with the absolute values of the both operands. In integer division (¥), the quotient is truncated to an integer. With the MOD operator, the remainder is given the sign of the dividend.
- The length of an expression is limited to 255 characters.

Relational Operators (=, < >, > <, <, >, = <, > =, = >, < =)

- Relational operations can be performed only when the operators are both strings or numeric values.
- In relational operations on strings, the codes are compared from the beginning of the string. If the length of the strings are different, the comparison is made according to the shorter string. The shorter string is said to be smaller if the results of this comparison are equal.
- When the result of a relational operation is true (the conditions are established), the result is −1. When the result is false (the conditions are not established), the result is expressed as 0.

Logical Operators (NOT, AND, OR, XOR)

Logical operation operands are truncated to integers and the result is calculated by performing the operation bit-by-bit.

Negation

X	NOT X
0	1
1	0

Logical product

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

Logical sum

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

Exclusive OR

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

String Operators (+)

- Strings may be concatenated using a + sign.
- The result of the operation (including intermediate results) may not exceed 255 characters.

Order of Precedence

1. (,)
2. Scientific function
3. Exponentiation
4. Sign (+, −)
5. *, /, ¥, MOD
6. Addition and subtraction
7. Relational operators
8. NOT
9. AND
10. OR, XOR

Operations are performed from left to right when order of precedence is identical.

1-6 TOUCH-KEYS

The 16 touch-keys on the screen are each assigned a character code value, so operation of these keys can be read by the INKEY$ statement.

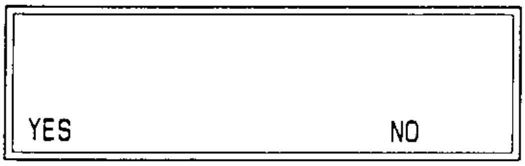
1 line {

① 240	② 241	③ 242	④ 243
⑤ 244	⑥ 245	⑦ 246	⑧ 247
⑨ 248	⑩ 249	⑪ 250	⑫ 251
⑬ 252	⑭ 253	⑮ 254	⑯ 255

8 columns

① through ⑯ are the numbers of the keys.
240 through 255 are the character codes (decimal) entered when these keys are pressed.

The program listed below assigns the words [YES] to key ⑬ and [NO] to key ⑯, and these words appear on the display at the appropriate locations. Pressing either one of these keys causes input of its character code value, and subsequent branching to the corresponding routine.



YES NO

```
10 CLS
20 LOCATE 0,3:PRINT "YES";
30 LOCATE 24,3:PRINT "NO";
40 A$=INKEY$
50 IF A$=CHR$(252) THEN 100
60 IF A$=CHR$(255) THEN 200
70 GOTO 40
100 'YES PROCESSING
200 'NO  PROCESSING
```

The routine noted below is very convenient when all 16 touch-keys are being used.

```
10 A$=INKEY$: A=ASC(A$)
20 IF A<240 THEN 10
30 ON A-239 GOTO Line number for ① key processing ,
                       Line number for ② key processing ,..........
                                   Line number for ⑯ key processing
```

1-7 PLOTTER COMMANDS

Various graphs can be drawn using the plotter-printer commands. The commands that are used with the plotter-printer are explained below.
The plotter-printer has two modes, a printer mode and a plotter mode.

Printer mode: Character output
Plotter mode: Line output

DIP Switches

There are 4 DIP switches on the printer. SW3 and SW4 are used for the following purposes.

SW3 CR/CR·LF mode
- OFF: Performs a carriage return (CR) when a CR code is sent.
- ON: Performs a CR and a line feed (LF) when a CR code is sent.

SW4 Scissoring ON/OFF
- OFF: Y-axis scissoring is turned off.
- ON: Y-axis scissoring is turned on.

Note: An LF code immediately following a CR is invalid in switching from the CR to the CR/LF mode.
(Also, an LF code is invalid if it follows immediately after a terminator in an ESCAPE sequence.)

- **Line Drawing Range when Scissoring is ON.**

Paper Width	100 mm	114 mm	A5	B5	A4
X direction	82 mm	96 mm	130 mm	164 mm	192 mm
−Y direction	120.2 mm	—	183.8 mm	229.8 mm	270.8 mm
+Y direction	6 mm	—	6 mm	6 mm	6 mm
Print columns (S1, 1)	34	40	54	68	80

Printer Mode

- Control codes (used in the character mode)

Backspace	08H
Paper feed	0AH
Back feed	0BH
Carriage return	0DH
Escape (ESC)	1BH

- Character Font (Escape codes are not used in the graphic mode.)

ESC + V (Terminator)	Set italics ON.
ESC + v (Terminator)	Set italics OFF.
ESC + E (Terminator)	Set boldface ON.
ESC + e (Terminator)	Set boldface OFF.
ESC + W (Terminator)	Set underline ON.
ESC + w (Terminator)	Set underline OFF.

Other graphic commands (excluding P and Q)

Plotter Mode

	Name	Command	Purpose
Graphic Commands	ORIGIN ○	O [absolute X coordinate, absolute Y coordinate] (Terminator)	Defines an origin of ORG coordinate.
	DRAW ○	D [starting X coordinate, starting Y coordinate] [, X coordinate, Y coordinate]* (Terminator) * At least one parameter must be present.	Draws straight lines connecting the points specified by ORG coordinates.
	RELATIVE DRAW ○	I X displacement, Y displacement [, X displacement, Y displacement] * (Terminator)	Draws straight lines connecting the points defined by the specified displacements in X and Y directions from the current pen position.
	MOVE ○	M X coordinate, Y coordinate (Terminator)	Moves the pen holder assembly with the pen up to the point defined by the specified ORG coordinates.
	RELATIVE MOVE ○	R X displacement, Y displacement (Terminator)	Moves the pen holder assembly with the pen up from the current pen position to the point defined by the specified X and Y displacements.
	QUAD ○	A starting X coordinate, starting Y coordinate, diagonal X coordinate, diagonal Y coordinate (Terminator)	Draws a quadrangle whose two diagonal points are defined by the two specified ORG coordinates and whose sides are parallel to the X and Y axes.
	CIRCLE ○	C [X center coordinate, Y center coordinate], radius [, initial arc angle, final arc angle] (Terminator) * final arc angle > initial arc angle	Draws a circle or circular arc around the center defined by the specified ORG coordinates. It draws an arc when the angle parameters are specified.
	AXIS ○	X axis direction, size of scale division, number of scale divisions (Terminator) * 0 ≦ axis direction < 4, size of scale division > 0, number of scale divisions > 0	Draws a coordinate axis in the +Y, +X, −Y or −X direction from the origin of ORG coordinate.
	GRID ○	G direction of stripes, range in X axis direction, range in Y axis direction [, stripe separation] (Terminator) * 0 ≦ direction of striped < 3, stripe separation > 0	Draws horizontal or vertical stripes from the current pen position within the specified range.
	LINE TYPE ○	L line type (Terminator) * 0 ≦ line type < 4	Specifies a line type which is a solid line, broken line, one-dot chained line or two-dot chained line.
	LINE SCALE ○	B line pitch (Terminator) * 0 ≦ broken line pitch < 1000	Specifies the pitch of a broken line, one-dot chained line, or two-dot chained line.
Character/Numeric Commands	ALPHA SCALE ○	S sx[, sy] (Terminator) * 0 ≦ character scale < 16	Specifies the size of characters and symbols to be printed.
	ALPHA ROTATE ○	Q rotational angle (Terminator) * 0 ≦ rotational angle (orientation) < 4	Specifies the rotational angle (orientation) of characters and symbols to be printed.

Name		Command	Purpose
Character/Numeric Commands	SPACE ○	Z spacing between current and next characters [, spacing between current and next lines] (Terminator)	Specifies the spacing between the current and next characters and/or the spacing between the current and next lines.
	YOKO ○	Y horizontal/vertical selection (Terminator) *0 ≦ horizontal/vertical selection < 2	Specifies whether subsequent character strings are to be printed horizontally or vertically.
	PRINT □	P character string (Terminator)	Allows the specified character strings or data to be printed while in the graphic mode.
	MARK ○	N mark number (Terminator) *0 ≦ mark number < 10	Draws the specified mark centered at the current pen position.
Control commands	NEW PEN ○	J color of pen (Terminator) *0 ≦ color of pen < 4	Specifies the color of pen: black, blue, green, red.
	LINE FEED ○	F number of lines (Terminator)	Feeds the paper the specified number of lines.
	HOME ○	H [distance from foremost drawing point] (Terminator) * distance from foremost drawing point ≧ 0.	Redefines the absolute coordinate system, or moves the pen holder assembly for inspection of the drawing.
	TEST ○	@ (Terminator)	Allows trial drawing or a check for proper inking.
Character control commands	TAB △	T number of print positions (Terminator)	Specifies a tab position.
	FORMAT △	? $\left\{\begin{matrix}0\\1\end{matrix}\right\}$ (Terminator)	Specifies a formatted program listing.

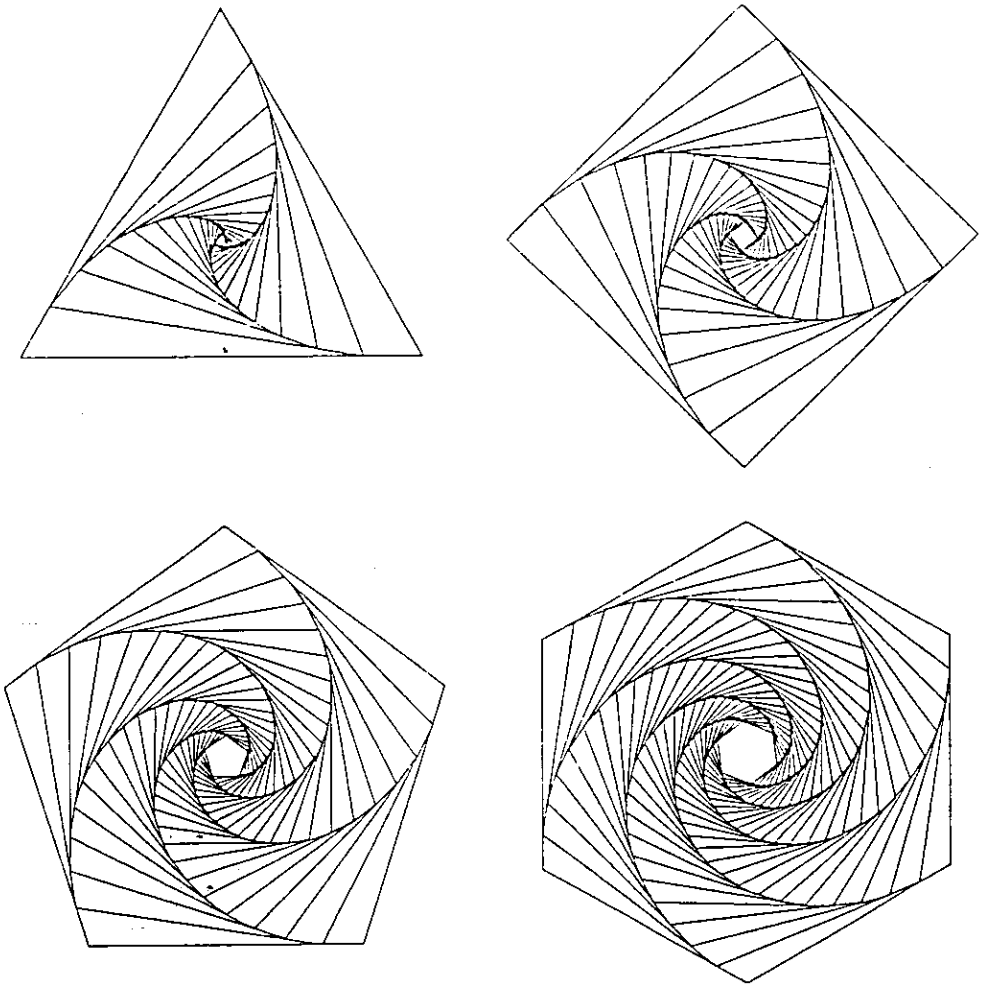
NOTE 1:

- An asterisk indicates that the term preceding it may appear more than once.
- Braces indicate that at least one of the parameters enclosed must be specified.
- Brackets indicate that the parameters enclosed may be omitted.
- All parameters are real numbers with up the 3-digit integers; any fractional part must be a multiple of 0.2 unless otherwise specified (i.e. range is −999.8 ~ 999.8).

NOTE 2:

- ○ indicates the command is effective in both the character and graphic modes.
- △ indicates the command is effective only in the character mode.
- □ indicates the command is effective only in the graphic mode.

Example of Printing Using the Plotter Mode



```
10 LPRINT CHR$(&H1C);CHR$(&H25)
20 DIM X(6),Y(6):KK=.15:R=43:ANGLE 0
30 FOR K=1 TO 4:LPRINT "J";K-1
40 OX=96*((K-1) MOD 2)+46:OY=-96*INT((K-1)/2)-46
50 LPRINT "O";OX;",";OY
60 FOR I=0 TO K+1
70 X(I)=R*SIN(360*I/(K+2)):Y(I)=R*COS(360*I/(K+2))
80 NEXT I
90 FOR I=1 TO 10+5*K
100 X(K+2)=X(0):Y(K+2)=Y(0)
110 FOR J=0 TO K+1
120 X1=ROUND(X(J),-2):Y1=ROUND(Y(J),-2)
130 X2=ROUND(X(J+1),-2):Y2=ROUND(Y(J+1),-2)
140 LPRINT "D";X1;",":Y1;",":X2:".":Y2
150 X(J)=X(J)+KK*(X(J+1)-X(J)):Y(J)=Y(J)+KK*(Y(J+1)-Y(J))
160 NEXT J:NEXT I:NEXT K:END
```

1-8 COMMUNICATIONS

RS-232C Interface

The RS-232C interface is the most widely used type of port for communications with other computers or other external devices.

- Input files, output files, and input/output files can be specified.
- Data are transferred to external devices via a communications line. This communication line is connected to the RS-232C port and data transfers are made over a duplex line by the start-stop (asynchronous) method.
- To prevent the loss of data that has been input, the data are also sent to a dedicated communications buffer and can be recalled with INPUT# statements.

File Descriptor

The file descriptor is input in the following format.

COM0 : [[Speed], [Parity], [Data], [Stop], [CS], [DS], [CD], [Busy], [Code]]
Example: COM0 : 2, E, 8, 1, N, N, N, B, N
COM0 : 2, E, 8, 2

a) Baud rate specification (Speed)

7 :	9600	(BPS)
6 :	4800	(BPS)
5 :	2400	(BPS)
4 :	1200	(BPS)
3 :	600	(BPS)
2 :	300	(BPS)
1 :	150	(BPS)
0 :	75	(BPS)

If the Baud rate is not specified, see PART 12 of "OWNER'S MANUAL" for setting the DIP switches for external devices.

b) Parity bit specification (Parity)

N :	No parity
E :	Even parity
O :	Odd parity

c) Character bit length specification (Data)

7 :	JIS 7 bits
8 :	JIS 8 bits

d) Stop bit specification (Stop)

1 :	1 bit
2 :	2 bits

e) CTS signal control (CS)

C : Controlled by CTS signal
N : CTS signal ignored

During CTS signal control, nothing is sent until CTS is ON.

f) DSR signal control (DS)

D : Controlled by DSR signal
N : DSR signal ignored.

During DSR signal control, an NR error is generated when data is received while DSR is OFF. Nothing is sent until DSR is ON.

g) CD signal control (CD)

C : Controlled by CD signal
N : CD signal is ignored.

During CD signal control, an NR error is generated if data is received while CD is OFF.

h) Buffer busy control (Busy)

B: Buffer busy control
N : No buffer busy control

An XOFF code (13H) is sent and data transfer from the host is temporarily halted when busy control is invoked and the empty area of the buffer is 64 characters or less. After the XOFF signal is sent, the data in the buffer is read. If there are 32 or fewer characters remaining in the buffer, the XON (11H) signal is sent and a send request is issued to the host. If the XOFF code is received from the host, data transfer is halted. Transfer is restarted when the XON code is sent.

i) System of input/output codes (Code)

S : SI/SO control
N : No SI/SO control

The character bit length during SI/SO control is 7 bits. To send a code that is greater than or equal to 80H, the SO code (0EH) is sent and operations remain in the SO mode until a code of 7FH is sent. To send a code equal to or less than 7FH in the SO mode, an SI code is sent and operations enter the SI mode. Character codes 80H ~ 9FH are processed as control codes in the SO mode.

- **Parameter default values (initial values)**

COM0	:	2, E, 8, 1, N, N, N, B, N
Speed	:	300 BPS
Parity	:	Even
Data bits	:	8
Stop bits	:	1
CS	:	Not checked
DS	:	Not checked
CD	:	Not checked
Busy	:	XON/XOFF control
Code	:	No SI/SO control.

Data Input/Output

The OPEN command is used for data input/output using the RS-232C interface. Processing is usually performed with sequential access files to facilitate data input/output. Random access files may also be opened, but such random access file commands such as GET and PUT may not be used.

```
  OPEN "COM0 : 5, E, 8, 2" AS #1
```

The SAVE and LOAD commands are used for input/output for program files.

```
SAVE "file descriptor"
LOAD "file descriptor"
```

Input and output of data files is performed using the [SAVE] and [LOAD] keys in the MENU mode. Exercise care when using the [LOAD] key because the data input is delayed until the BRK key is pressed.

List of BASIC Commands

The BASIC commands and functions related to the RS-232C port are shown below.

Command	Purpose
OPEN	Declares that communications line in use.
CLOSE	Closes communications line that was open.
PRINT#	Outputs data to communications line.
PRINT# USING	Outputs data to communications line.
INPUT#	Reads data from communications line.
LINE INPUT#	Reads data from communications line.
INPUT$	Reads data from communications line.
EOF	Function indicating receive buffer status.
LOF	Function indicating number of bytes remaining in receive buffer.
SAVE	Outputs program to communications line.
LOAD	Reads program from communications line.
MERGE	Reads and merges program from communication line with program in memory.

1-9 DATA FILES

Data files —— Sequential files
└── Random files

Sequential Files

Inputting or outputting data in sequence starting from the first item in the file is referred to as sequential processing.

- **File Specification**
  Files for input or output are specified by the OPEN command.
  OPEN "file descriptor" FOR INPUT AS # file number (Input specification)
  OPEN "file descriptor" FOR OUTPUT AS # file number (Output specification)
  OPEN "file descriptor" FOR APPEND AS # file number (Append specification)

- **Input and Output Commands**
  BASIC provides input and output commands for both input and output files, and a function that detects the end of a file (EOF) is also included.

Command	Input Specification	Output Specification	Append Specification
PRINT #	×	○	○
PRINT # USING	×	○	○
INPUT #	○	×	×
LINE INPUT #	○	×	×
INPUT $	○	×	×
EOF	○	—	—

○ : Execution possible
× : Execution impossible

- **Data Configuration**
  Data are arranged in variable-length records. Delimiters can be CR codes (0DH), commas (2CH), spaces (20H), double quotation marks (22H), or CR · LF codes (0DH · 0AH). Spaces (20H) may also be used as delimiters for numeric variables. A SUB code (1AH) is written at the end of the file.

Random Files

Input and output that is performed without regard to sequence is called random access processing.

- With random files, records specified by record numbers are processed, and an OPEN statement is used to process the file as an input/output file.

  OPEN "file descriptor" AS # file number

- The GET and PUT statements are used for data input and output commands. The GET and PUT statements are used to input data to and output data from the input/output buffer created by the OPEN statement.
- The FIELD statement must be used to assign a string variable to the input/output buffer to use the data in the buffer.
- The LSET and RSET statements are used for assigning variable strings allocated to the input/output buffer.

- The size of the current file is shown by the LOF function.
- Processing of random files when the data is in fixed-length records (256 characters maximum) is performed sequentially, and data delimiters are not used.

The method for entering statements is explained below.

- Words in bold type are commands or functions, and they must be entered as shown.
- Braces indicate that one of the parameters enclosed must be specified.
- Commas contained in braces must be written in the position shown.
- Brackets indicate that the parameters enclosed may be omitted. Brackets themselves are not entered.
- An asterisk indicates that the term preceding it may appear more than once.
- Numeric expressions—Constants, expressions, and numeric variables (e.g. 10, 10 + 25, A, unit cost ∗ quantity)
- String expressions—String constants, string variables, and string expressions (e.g. "ABC", A$, and A$ + B$)
- Expressions—General term for numeric and string expressions
- Arguments—Elements used by commands and functions

Example: MID$ function

$$\text{MID\$}\quad \underset{\text{String expression}}{\underline{\text{(string array,}}}\quad \underset{\text{Numeric expression}}{\underline{\text{position}}}\quad \underset{\text{Numeric expression}}{\underline{\text{[, number of characters])}}}$$

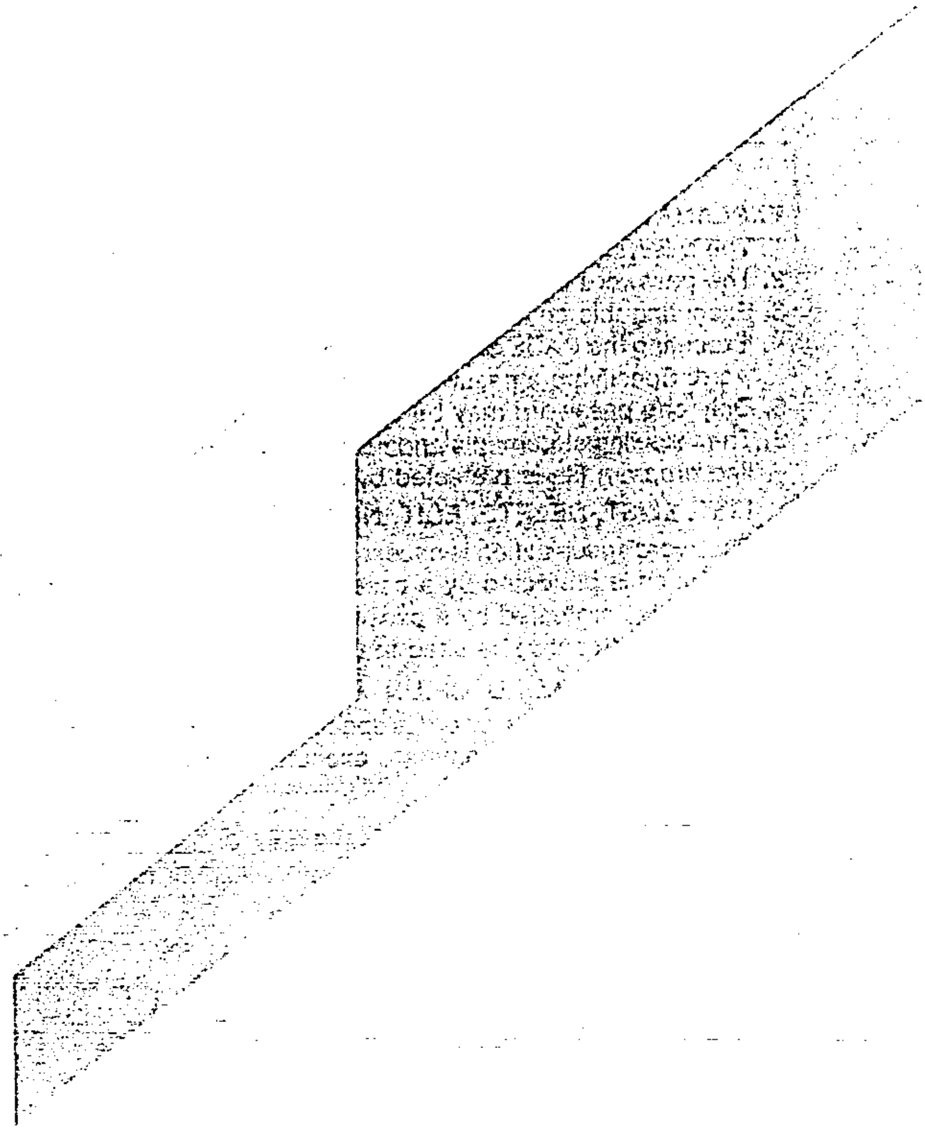
The term "string expression" under "string array" describes that array. Likewise, "numeric expression" under "position" and "numeric expression" under "number of characters" are descriptors. Also, since the comma and number of characters are enclosed in brackets, they may be omitted.

Example: GOSUB Statement

$$\text{GOSUB}\left\{\begin{array}{c}\dfrac{\text{Branch line number}}{\text{Line number}}\\ \\ \dfrac{\text{Program filename}}{\text{String expression}}\end{array}\right\}$$

This example illustrates two descriptors for GOSUB: the line number of the subroutine to which the program branches and filename to which the program branches.

BASIC COMMANDS



MANUAL COMMANDS

PASS

PURPOSE: Specifies or cancels a password.

FORMAT: PASS $\underbrace{\text{("password")}}_{\text{String expression}}$

EXAMPLE: PASS "TEXT"

PARAMETERS:
1. The specified password is registered for the specified file.
2. The password must be a string of 1 ~ 8 characters.
3. All characters after the first 8 are ignored when 9 or more characters are entered.

EXPLANATION:
1. The password is used to protect a program file.
2. The password is registered in the BASIC programming mode.
3. Executing this command registers a password when no password previously exists.
4. Executing the PASS statement using a previously registered password cancels the password. Specifying a password that is different from that registered, results in a PR error.
5. Only one password may be used for each file.
6. The following statements and commands may not be executed when the currently specified program file is protected by a password.
   LIST, LLIST, DELETE, EDIT, NEW
7. Password specification is necessary when the SAVE command is executed for a program file which is protected by a password.
8. A program protected by a password cannot be saved in ASCII format.
9. A password cannot be used in the CAL mode.

SEE: LOAD, CHAIN, MERGE
A password is specified for the external files when these commands and statements are executed.

NEW

PURPOSE: Deletes a program.

EXAMPLE: NEW

EXPLANATION:

1. Deletes the currently specified program.
2. "Ready" is displayed on the screen after the program is deleted, and the computer stands by for command input.
3. All files that are currently opened are closed.
4. This command cannot be executed for program files that are protected by a password.
5. This command cannot be used in the CAL mode.

SYSTEM

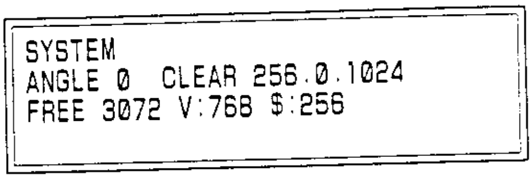
PURPOSE: Indicates the CLEAR and ANGLE settings and the free area for work.

EXAMPLE: SYSTEM

EXPLANATION:
Indicates the CLEAR and ANGLE settings and the free area available for text, numeric variables, and string variables.

SEE: Memory map

SAMPLE EXECUTION: SYSTEM EXE



```
SYSTEM
ANGLE 0  CLEAR 256.0.1024
FREE 3072 V:768 $:256
```

Values shown for FREE, V, and $ are free areas for text, numeric variables, and character variables, respectively.

CLEAR

PURPOSE: Clears all variables and determines the amount of space available to BASIC according to the parameter entered. Also closes any files that are open.

FORMAT: CLEAR [character area size] [, machine language area size] [, system work area size]

EXAMPLE: CLEAR 500, 100, 1000

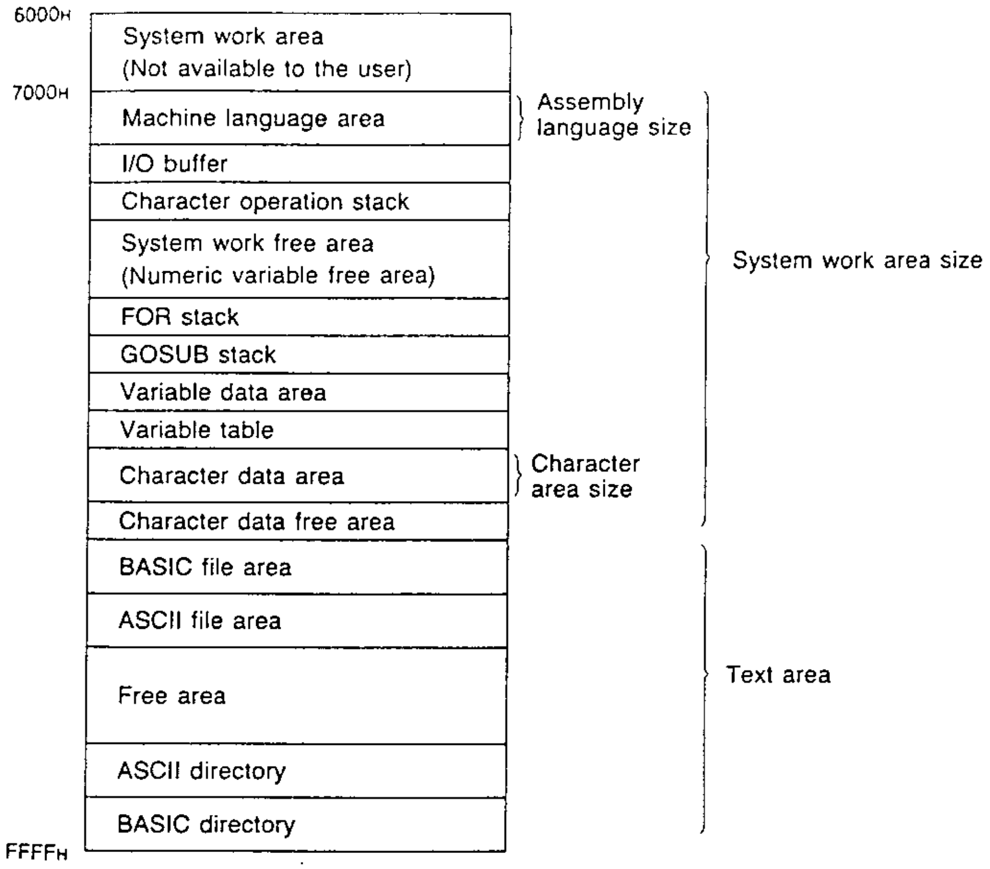
PARAMETERS: 1. character area size: Numeric expression

Determines the size of the area into which text data can be entered. When the NEWALL command is executed, the size that is set aside varies according to the memory capacity.

Memory capacity (bytes)	Character area size (bytes)
8K	256
40K	1K (=1024)

2. machine language area size: Numeric expression (<4K)
   Determines the size of the area available for machine language programs. When NEWALL is executed, 0 byte is specified.
3. system work area size: Numeric expression $\left(\begin{array}{l}\leq 4\text{KB for 8KB RAM}\\ <36\text{KB for 40KB RAM}\end{array}\right)$
   Determines the size of the system work area. When NEWALL is executed, 1/4 of memory capacity is specified (1024 for 8KB RAM, 9215 bytes for 40KB RAM). See the memory map for details on the system work area.
4. The size of the system work area cannot be set during program execution.
5. The total size of the character area and the machine language area must be less than the system work area size.

Memory Map


6000H
System work area
(Not available to the user)
7000H
Machine language area
I/O buffer
Character operation stack
System work free area
(Numeric variable free area)
FOR stack
GOSUB stack
Variable data area
Variable table
Character data area
Character data free area
BASIC file area
ASCII file area
Free area
ASCII directory
BASIC directory
FFFFH
Assembly language size
System work area size
Character area size
Text area

MON

PURPOSE: Activates the monitor.

EXPLANATION:

1. Activates the monitor. See the explanation of the monitor mode on page 192.
2. The ">" prompt appears on the screen and the computer stands by for command input when MON is entered and the EXE key is pressed.
3. The computer returns to the MENU mode when MENU is pressed, and to the CAL mode when CAL is pressed.

DELETE

PURPOSE: Deletes program lines.

FORMAT: $\text{DELETE } \underset{\text{Line number}}{\underline{\text{start line number}}} \text{ [(-) } \underset{\text{Line number}}{\underline{\text{[end line number]]}}}$

EXAMPLE: DELETE 50 - 100

PARAMETERS:
1. start line number: Integer in the range of 1 $\leqq$ line number $\leqq$ 65535
2. end line number: Integer in the range of 1 $\leqq$ line number $\leqq$ 65535

EXPLANATION:

1. Deletes program lines within the specified range.
2. A minus sign is used as the delimiter between the start line number and the end line number.
3. The four examples below illustrate specification of the delete range.
   a) DELETE100 EXE (Line 100 is deleted.)
   b) DELETE100 - 200 EXE (Lines 100 through 200 are deleted.)
   c) DELETE200 - EXE (Lines from 200 through the end of the program are deleted.)
   d) DELETE - 50 EXE (All lines from the beginning of the program through line 50 are deleted.)
4. The start line number must be less than or equal to the end line number.
5. All lines above the start line number are deleted when the specified start line does not exist. Similarly, all lines below the end line number are deleted when the specified end line number does not exist.
6. The computer stands by for command input when deletion is complete.
7. This command cannot be used when the currently specified program file is protected by a password.
8. This command closes all files that are open.
9. This command cannot be used in the CAL mode.

LIST

PURPOSE: Displays all or a part of the currently specified program.

FORMAT:

```
      ( [start line number] [ - [end line number]] )
LIST  {   Line number               Line number    }
      ( [ . ]                                      )
```

EXAMPLE:

LIST 100
LIST 100 - 300
LIST - 400
LIST

PARAMETERS:

1. start line number: Integer in the range of 1 ≦ line number ≦ 65535
2. end line number: Integer in the range of 1 ≦ line number ≦ 65535

EXPLANATION:

1. Displays the currently specified program in the range specified by the line numbers.
2. A minus sign must be used as the delimiter between line numbers.
3. The following five examples illustrate specification of the display range.
   a) LIST EXE (All lines from beginning of program)
   b) LIST 30 EXE (Line 30)
   c) LIST 50 - 100 EXE (Lines 50 through 100)
   d) LIST 200 - EXE (From line 200 through end of program)
   e) LIST - 80 EXE (From beginning of program through line 80)
4. Using a period in place of the line number displays the most recently handled (i.e. written, edited, executed). If a program is halted during execution by an error, executing "LIST ." displays the line in which the error was generated.
5. When the specified start line number does not exist, the first line number above that specified is taken as the start line number.
6. When the specified end line number does not exist, the greatest line number not exceeding that specified is taken as the end line.
7. The start line number must be smaller than the end line number.
8. LIST command execution can be halted by pressing the BRK key.
9. Press the STOP key to momentarily halt LIST command execution. To restart execution, press the EXE key or one of the alphanumeric keys.
10. The computer stands by for command input after the program list is displayed.
11. This command cannot be used if the currently specified program file is protected by a password.
12. This command cannot be used in the CAL mode.

SEE: EDIT, VARLIST

EDIT

PURPOSE: Enters the BASIC edit mode.

FORMAT:

EDIT { [start line number] / Line number or period [.] }

EXAMPLE: EDIT 100

PARAMETERS: start line number: Integer in the range of 1 ≦ line number ≦ 65535

EXPLANATION: .

1. Enters the BASIC edit mode and displays the program from the specified line number. The cursor is displayed and editing becomes possible when either the ◀ or ▶ key is pressed.
2. Using a period in place of the line number displays the most recently handled (i.e. written, edited, executed). If a program is halted during execution by an error, executing "EDIT ." displays the line in which the error was generated.
3. When the specified start line number does not exist, the first line number above that specified is taken as the start line number.
4. This command cannot be used if the currently specified program file is protected by a password.
5. This command cannot be used in the CAL mode.

SEE: LIST

VARLIST

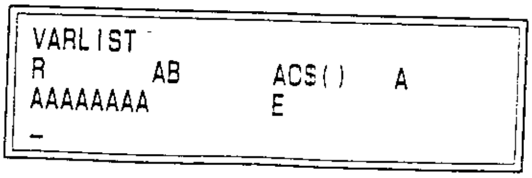
PURPOSE: Displays variable names and array names.

EXAMPLE: VARLIST

EXPLANATION:

1. Displays all currently existing variable names and array names.
2. VARLIST command execution can be halted by pressing the BRK key.
3. Press the STOP key to momentarily halt VARLIST command execution. To restart execution, press the EXE key or one of the alphanumeric keys.
4. The computer stands by for command input after the variable list is displayed.

SAMPLE EXECUTION: VARLIST EXE



```
VARLIST
R        AB         ACS( )    A
AAAAAAAA            E
_
```

RUN

PURPOSE: Executes a program.

FORMAT: RUN [execution start line]
Line number

EXAMPLE:

```
RUN
RUN 100
```

PARAMETERS: start line number: Integer in the range of 1 ≦ line number ≦ 65535

EXPLANATION:

1. Execution starts from the beginning of the program when the line number is omitted.
2. When the specified start line number does not exist, the first line number above that specified is taken as the start line number.
3. This command closes all files that are open.
4. Variable and array values are not cleared.
5. This command cannot be used within a program.
6. This command cannot be used in the CAL mode.

TRON

PURPOSE: Specifies the trace mode.

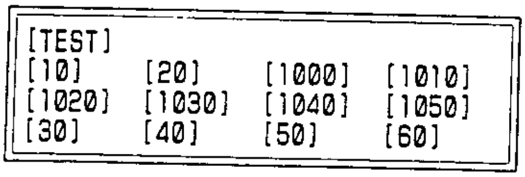
EXAMPLE: TRON

EXPLANATION:

1. This command specifies the trace mode. The filenames and line numbers corresponding to subsequently executed files and lines are displayed enclosed in brackets.
2. Filenames are displayed at the beginning of program execution or when the program file is changed by GOTO "filename", etc. Line numbers are displayed each time a line is executed.
3. The program stays in the TRON mode until the TROFF statement is executed.

SEE: TROFF

SAMPLE EXECUTION: RUN EXE



```
[TEST]
[10]    [20]    [1000]  [1010]
[1020]  [1030]  [1040]  [1050]
[30]    [40]    [50]    [60]
```

TROFF

PURPOSE: Cancels the trace mode.

EXPLANATION:

Cancels the trace mode (entered using the TRON statement).

SEE: TRON

FUNDAMENTAL COMMANDS

END

PURPOSE: Terminates program execution.

.XAMPLE: END

EXPLANATION:

1. Terminates program execution, and the computer stands by for command input.
2. Closes all files that are open.
3. Variables and arrays are not cleared.
4. Any number of END statements can be used in a single program. Program execution is terminated and open files are closed automatically at the end of the program even if an END statement is not included.
5. When files are closed, an OM error is generated for each of the files that cannot be closed due to insufficient memory capacity.
6. When ON ERROR GOTO is specified, error branching is not performed even if the OM error message is displayed. The CLOSE statement should be included before the END statement in error handling routines.

SAMPLE PROGRAM:

```
10 FOR I=1 TO 20
20 IF I>10 THEN END
30 PRINT I;
40 NEXT I
```

STOP

PURPOSE: Temporarily halts program execution.

EXAMPLE: STOP

EXPLANATION:

1. Temporarily halts execution of a program, and the computer stands by for command input. The CONT key is pressed while holding down the SHIFT key to resume program execution.
2. Open files, variable values and array values are retained as they are at the point when execution is halted.
3. The STOP status is canceled when an error is generated, the mode is changed, or the program is edited while program execution is halted.

SEE: CONT

SAMPLE PROGRAM:

```
10 FOR I=1 TO 10
20 IF I=6 THEN STOP:PRINT
30 PRINT I;
40 NEXT I
```

GOTO

PURPOSE: Branches unconditionally to a specified branch destination.

FORMAT:

```
      { branch destination line number }
      { ------------------------------ }
      {          Line number           }
GOTO  {                                }
      {       "program filename"       }
      {       ------------------       }
      {       String expression        }
```

SAMPLE:

```
GOTO  1000
GOTO  "DEMO1"
```

PARAMETERS:

1. branch destination line number: Integer in the range of 1 ≦ line number ≦ 65535
2. program filename: String expression.

EXPLANATION:

1. Specifying a line number causes program execution to jump to that line number in the current program.
2. Specifying a program filename causes program execution to jump to the first line number of the specified program file.
3. A UL error is generated when the specified line number does not exist, while an NF error is generated when the specified filename does not exist.

SAMPLE PROGRAM:

```
10 PRINT "PRESS [STOP]"
20 PRINT "TO HALT EXECUTION"
30 PRINT.
40 FOR I=1 TO 30:NEXT I
50 GOTO 10
```

GOSUB

PURPOSE: Jumps to a specified subroutine.

FORMAT:

GOSUB { branch destination line number / Line number ; "program filename" / String expression }

EXAMPLE: GOSUB 100
GOSUB "SAMPLE"

PARAMETERS:

1. branch destination line number: Integer in the range of 1 ≦ line number ≦ 65535
2. program filename: String expression.

EXPLANATION:

1. Program execution branches to the subroutine that starts at the specified line number. Execution is returned from the subroutine by the RETURN statement.
2. Nested subroutines that call another subroutine (from within the present subroutine) are possible to the extent allowed by the memory stack capacity (see Memory map). Exceeding the memory stack capacity results in an OM error.
3. A UL error is generated when the specified line number does not exist, while an NF error is generated when the specified filename does not exist.
4. A GS error is generated when the CLEAR statement is used within a subroutine.

SEE: RETURN

SAMPLE PROGRAM:

```
10 REM***MAIN***
20 GOSUB 40
30 END
40 REM***SUBROUTINE 1***
50 PRINT"SUBROUTINE 1";
60 GOSUB 80
70 RETURN
80 REM***SUBROUTINE 2***
90 PRINT"SUBROUTINE 2"
100 RETURN
```

RETURN

PURPOSE: Returns execution from a subroutine to the main program.

FORMAT:

$$\text{RETURN}\left[\left\{\begin{array}{c}\underline{\text{Return line number}}\\ \text{Line number}\\ \underline{\text{"program filename"}}\\ \text{String expression}\end{array}\right\}\right]$$

EXAMPLE:

```
RETURN
RETURN  30
```

PARAMETERS: 1. line number: Integer in the range of 1 ≦ line number ≦ 65535
2. program filename: String expression.

EXPLANATION:

1. Returns execution to the specified line number or program filename.
2. The default option is line following that which called the subroutine when the line number specification is omitted.
3. Specifying a line number returns execution to the specified line in the current program file. If the subroutine has been called from another program file, execution returns to the specified line number of the subroutine.
4. When a program filename is specified, execution returns to the beginning of the specified program file.
5. A GS error is generated when the RETURN statement is executed without first executing a GOSUB statement.

SEE: GOSUB, ON~GOSUB

SAMPLE PROGRAM:

```
10 REM SUBROUTINE
20 GOSUB 100
30 END
100 PRINT"SUBROUTINE 1"
110 GOSUB 200
120 RETURN
200 PRINT"SUBROUTINE 2"
210 RETURN
```

ON GOTO

PURPOSE: Jumps to a specified branch destination in accordance with a specified branching condition.

FORMAT: ON $\underset{\text{Numeric expression}}{\underline{\text{condition}}}$ GOTO [branch destination] [, [branch destination]]*

Branch destination: $\left\{\begin{array}{c} \underline{\text{destination branch line number}} \\ \text{line number} \\ \underline{\text{program file name}} \\ \text{string expression} \end{array}\right\}$

EXAMPLE: ON A GOTO 100, 200, 300

PARAMETERS:
1. branch condition: Numeric expression truncated to an integer
2. line number: Integer in the range of 1 ≦ line number ≦ 65535
3. program filename: String expression.

EXPLANATION:
1. The GOTO statement is executed in accordance with the value of the expression used for the branch condition. For example, execution jumps to the first branch destination specified when the value is 1, to the second destination when the value is 2, etc.
2. Program execution does not branch and execution proceeds to the next statement when the value of the branch condition is less than 1, or if a branch destination corresponding to that value does not exist.
3. Up to 99 branch destinations may be specified.
4. Line numbers and program filenames cannot be mixed as branch destinations.

SAMPLE PROGRAM:

```
10 INPUT"1 OR 2":A
20 ON A GOTO 40,50
30 END
40 PRINT"ONE":END
50 PRINT"TWO":END
```

Execution jumps to line 40 if 1 EXE is entered or to line 50 if 2 EXE is entered. Otherwise, execution terminates at line 30.

ON GOSUB

PURPOSE: Jumps to a specified subroutine in accordance with a specified branching condition.

FORMAT: ON $\frac{\text{condition}}{\text{Numeric expression}}$ GOSUB [branch destination] [, [branch destination]]*

Branch destination: $\left\{ \begin{array}{l} \frac{\text{destination branch line number}}{\text{line number}} \\ \frac{\text{program file name}}{\text{string expression}} \end{array} \right\}$

EXAMPLE: ON A GOSUB 1000, 1100, 1200

PARAMETERS:
1. branch condition: Numeric expression truncated to an integer
2. line number: Integer in the range of 1 ≦ line number ≦ 65535
3. program filename: String expression.

EXPLANATION:
1. The GOSUB statement is executed in accordance with the value of the expression used for the branch condition. For example, execution jumps to the first branch destination specified when the value is 1, to the second destination when the value is 2, etc.
2. Program execution does not branch and execution proceeds to the next statement when the value of the branch condition is less than 1, or if a branch destination corresponding to that value does not exist.
3. Up to 99 branch destinations may be specified.
4. Line numbers and program filenames cannot be mixed as branch destinations.

SEE: RETURN

SAMPLE PROGRAM:

```
10 S1=0:S2=0
20 FOR I=1 TO 100
30 ON (I MOD 2)+1 GOSUB 1000,1100
40 NEXT I
50 PRINT "S1=";S1
60 PRINT "S2=";S2
70 END
1000 S1=S1+I:RETURN
1100 S2=S2+I:RETURN
```

S1 calculates sum of even numbers from 1 to 100, S2 calculates sum of odd numbers from 1 to 100.

IF ~ THEN ~ ELSE/IF ~ GOTO ~ ELSE

PURPOSE: Executes the THEN statement or GOTO statement when the specified condition is met. The ELSE statement is executed when the specified condition is not met.

FORMAT:

```
        condition      ⎧ THEN   statement          ⎫        ⎧  statement        ⎫
   IF  ───────────    ⎨                 [ : statement] ⎬ ELSE ⎨    [ : statement] ⎬
        Numeric        ⎩ GOTO   branch destination ⎭        ⎩ branch destination ⎭
        expression
```

```
                                  ⎧ destination branch line number ⎫
                                  ⎪ ──────────────────────────────  ⎪
        Branch destination:       ⎨              line number        ⎬
                                  ⎪ program filename                ⎪
                                  ⎪ ────────────────                ⎪
                                  ⎩ string expression               ⎭
```

EXAMPLE:

IF A=0 THEN 300 ELSE 400
IF K$="Y" THEN PRINT X ELSE PRINT Y

PARAMETERS:

1. branch condition: Numeric expression truncated to an integer
2. line number: Integer in the range of 1 ≦line number≦65535
3. program filename: String expression.

EXPLANATION:

1. The statement following the THEN clause is executed, or execution jumps to the destination specified by the GOTO statement when the branch condition is met.
2. If the branch condition is not met, the statement following the ELSE statement is executed, or the program jumps to the specified branch destination. Execution proceeds to the next program line when the ELSE statement is omitted.
3. The format "IF A THEN ~ " results in the condition being met when value of the expression (A) is not 0 (absolute value of A < 1 × 10^{-99}). The condition is not met when the value of the expression is 0.
4. IF statements can be nested (an IF statement may contain other IF statements). In this case, the THEN ~ ELSE statements are related by their proximity. The GOTO ~ ELSE combinations have the same relationships.
   IF ~ THEN IF THEN ~ ELSE IF ~ THEN ~ ELSE ~ ELSE ~

SAMPLE PROGRAM:

```
10 INPUT "1 TO 9";A
20 IF (0<A)AND(A<10) THEN PRINT "GOOD!"ELSE 10
```

"GOOD" displayed for input values from 1 to 9. Re-input is requested for other values.

FOR ~ NEXT

PURPOSE: Executes the program lines between the FOR statement and NEXT statement and increments the control variable, starting with the initial value. Execution is terminated when value of the control variable exceeds the specified final value.

FORMAT: FOR control variable name = $\underbrace{\text{initial value}}_{\text{Numeric expression}}$

TO $\underbrace{\text{final value}}_{\text{Numeric expression}}$ [STEP $\underbrace{\text{increment}}_{\text{Numeric expression}}$]

NEXT [Control variable name] [, Control variable name]*

EXAMPLE:

```
FOR I=1  TO  10  STEP  0.1
≀
NEXT I
```

PARAMETERS:
1. control variable name: Array variables cannot be used.
2. initial value: Numeric expression
3. final value: Numeric expression
4. increment: Numeric expression (default value = 1)

EXPLANATION:

1. None of the statements between FOR and NEXT are executed and the program proceeds to the next executable statement after NEXT when the initial value is greater than the final value.
2. Each FOR requires a corresponding NEXT.
3. FOR ~ NEXT loops can be nested (a FOR ~ NEXT loop can be placed inside another FOR ~ NEXT loop). Nested loops must be structured as shown below with NEXT appearing in reverse sequence of the FOR (e.g. FOR A, FOR B, FOR C ~ NEXT C, NEXT B, NEXT A).

```
┌10  FOR I=1 TO 12 STEP 3
│┌20  FOR J=1 TO 4 STEP 0.5
││30  PRINT I, J
│└40  NEXT J
└50  NEXT I
 60  END
```

4. Up to 255 FOR ~ NEXT loops may be nested (subject to memory capacity limitations).
5. The control variable may be omitted from NEXT. However, use of the control variable in the NEXT statement is recommended when using nested loops.

6. NEXT statements can be chained by including them under one NEXT statement, separated by commas.

```
┌10 FOR I=1 TO 12 STEP 3        ┌10 FOR I=1 TO 12 STEP 3
│┌20 FOR J=1 TO 4 STEP 0.5      │┌20 FOR J=1 TO 4 STEP 0.5
││30 PRINT I,J                   ││30 PRINT I,J
│└40 NEXT J                     └┴40 NEXT J,I
└─50 NEXT I                        50 END
  60 END
```

7. The control variable retains the value which exceeds the final value (and terminates the loop) when loop execution is complete. With the loop FOR I = 3 TO 10 STEP 3, for example, the value of control variable I is 12 when execution of the loop is complete.
8. Jumping out of a FOR ~ NEXT loop is also possible. In this case, the current control variable value is retained in memory, and the loop can be resumed by returning with a GOTO statement.

REM(')

PURPOSE: Allows remarks or comments to be included within a program. This command is not executed.

FORMAT:

```
{REM}  comments
{ ' }  String expression
```

EXAMPLE: REM or '

PARAMETERS: comments: String expression

EXPLANATION:

1. Including an apostrophe or REM statement following the line number indicates that the following text is comments and should be ignored in program execution.
2. The apostrophe may be included at the end of any executable statement to indicate that the following text is comments. The REM statement can only be used at the beginning of a line.
3. Any command following the REM statement is treated as a comment and is not executed.

```
PRINT A:  REM 123   OK
               Comments
PRINT A  REM 123   error
PRINT A ' 123      OK
          Comments
```

4. An apostrophe is entered by pressing the = key while holding down the SHIFT key.

SAMPLE PROGRAM:

```
10 ' REM(') indicates comment
```

LET

PURPOSE: Assigns the value of an expression on the right side of an equation to the variable on the left side.

FORMAT: [LET] numeric variable name = Numeric expression
[LET] string variable name = String expression

EXAMPLE: LET A = 15
LET K$ = "123"

EXPLANATION:

1. Assigns the value of an expression on the right side of an equation to the variable on the left side.
2. Numeric expressions can only be assigned to numeric variables, and string expressions can only be assigned to string variables. A TM error is generated when an attempt is made to assign a string expression to a numeric variable, and vice versa.
3. LET may be omitted.

SAMPLE PROGRAM:

```
10 LET A=10
20 B=20
30 PRINT A;B
```

DATA

PURPOSE: Holds data for reading by the READ statement.

FORMAT: $\text{DATA } \underbrace{[\text{data}]}_{\text{Constant}} \; \underbrace{[,\ [\text{data}]]^{*}}_{\text{Constant}}$

EXAMPLE: DATA 10, 5, 8, 3
DATA CAT, DOG, LION

PARAMETERS:
1. data: String constants or numeric constants
2. string constants: Quotation marks are not required unless the string contains a comma which is part of the data. A null data string (length 0) is assumed when data is omitted from this statement.

EXPLANATION:
1. This statement can be used anywhere in the program to hold data to be read by the READ command.
2. Multiple data items are separated by commas.

SEE: READ, RESTORE

SAMPLE PROGRAM:

```
10 READ A$
20 RESTORE 60
30 READ B$
40 PRINT A$+" "+B$
50 DATA AD 1990,NO USE
60 DATA ABCDEF
```

READ

PURPOSE: Reads the contents of the DATA statement into memory.

FORMAT: READ Variable name [,Variable name]*

EXAMPLE: READ A, B
READ C$, X, Y

PARAMETERS: Variable name

EXPLANATION:

1. Assigns the data contained in a DATA statement to the variables on a one-by-one basis.
2. Numeric data can only be assigned to numeric variables, and string data can only be assigned to string variables. A TM error is generated when an attempt is made to assign string data to a numeric variable, and vice versa.
3. The data in DATA statements is read from the lowest line number in ascending order. Data are read in order from the beginning of a DATA statement.
4. The first execution of the READ statement reads the first data item contained in the first DATA statement. Subsequent executions read data items in sequential order.
5. The data line to be read can be specified using the RESTORE statement.

SEE: DATA, RESTORE

SAMPLE PROGRAM:

```
10 READ X
20 IF X<>0 THEN PRINT X;:GOTO 10
30 END
100 DATA 1,2,3,4,5,6,7,8,9
110 DATA 9,8,7,6,5,4,3,2,1
120 DATA 0
```

RESTORE

PURPOSE: Specifies a DATA line for reading by the READ statement.

FORMAT: RESTORE [line number] / (Numeric expression)

EXAMPLES:
RESTORE
RESTORE 1000
RESTORE (10∗10)
└─line 100

PARAMETERS: line number: Integer in the range of 1 ≦ line number ≦ 65535

EXPLANATION:

1. The first DATA line in the program file containing the READ statement is the default option when the line number is omitted.
2. When a line number is specified, the first data item in the specified DATA line is read by the next READ statement execution. A UL error is generated when the specified line number does not exist, while a DA error is generated when no data exist in the specified line.
3. A numeric expression can be used for line number specification. In this case, the numeric expression must be enclosed in parentheses.

SEE: READ, DATA

SAMPLE PROGRAM:

```
10 READ X
20 IF X<>0 THEN PRINT X;:GOTO 10
30 RESTORE 110
40 READ X
50 IF X<>0 THEN PRINT X;:GOTO 40
60 END
100 DATA 1,2,3,4,5,6,7,8,9
110 DATA 9,8,7,6,5,4,3,2,1
120 DATA 0
```

PRINT

PURPOSE: Displays data on the screen.

FORMAT: PRINT [output data] $\left\{\begin{matrix} ; \\ , \end{matrix}\right\}$ [output data]*

Output data: TAB (Numeric expression), REV, NORM, numeric expression, string array

EXAMPLE: PRINT "AD1990"
PRINT REV; "ABCDE"

PARAMETERS: output data: Output control function, numeric expression, or string expression

EXPLANATION:

1. Output of a numeric or string expression displays the value or string on the screen. Control function output results in the operation determined by the function being performed.
2. Numeric expressions are displayed in decimal notation with values longer than 10 digits being displayed using a mantissa rounded off to 10 digits, plus a 2-digit exponent.
   a) Integers: Values less than 1 × 10^{10}
   b) Fraction: Decimal fractions smaller than 10 digits
   c) Exponent: Other values
   A space is added after displayed numeric expressions, with negative expressions preceded by a minus sign, and positive expressions preceded by a space. Expressions are displayed as integers, fractions, or exponential expressions, with the display format automatically selected according to the value of the expression.
3. String expressions are displayed unchanged. There are, however, special operations for internal codes 00_H ~ $1F_H$, $7F_H$ (see CHARACTER CODE TABLE on page 196). Internal codes $F0_H$~FF_H can be used to freely specify the shape of characters using the DEFCHR$ statement.
4. Output is displayed on the screen from the current position of the cursor to the right. A line feed results when the cursor reaches the last column on the last line of the screen (lower right), scrolling the entire screen upwards. Subsequent output is displayed from the beginning of the bottom line of the screen (lower left).
5. Separating expressions with commas causes each output to be followed by a line change.
6. Separating expressions with semicolons causes each output to be displayed immediately following the previous output.
7. Including a semicolon at the end of this statement causes the cursor to remain at position immediately following the displayed output.

SEE: TAB, REV, NORM

SAMPLE PROGRAM:

```
10 PRINT "PRINT DISPLAYS MESSAGES"
20 PRINT "ON THE SCREEN"
```

TAB

PURPOSE: Outputs a horizontal tab specification to the screen or printer.

FORMAT: TAB $\underline{\text{(tab specification)}}$

Numeric expression

EXAMPLE: PRINT TAB (5) ; "ABC"

PARAMETERS: tab specification: Numeric expression truncated to an integer in the range of 0 ≦ tab specification < 256.

EXPLANATION:

1. Used in the PRJNT, LPRINT, and PRINT# statements to specify a display position on a line. Spaces are inserted from the left end of the line to the specified position.
2. The display position is determined by counting from the left end of the line (position 0 and) to the right, up to the specified value.
3. A tab specification value in an LPRINT statement which is less than the current printhead position causes the tabulation to performed following a carrier return/line feed.

SEE: PRINT, LPRINT, PRINT#

SAMPLE PROGRAM:

```
10 FOR I=0 TO 25
20 PRINT TAB(I);"ABCDEFG"
30 FOR J=0 TO 25 : NEXT J
40 NEXT I
```

REV

PURPOSE: Displays characters in reverse field.

FORMAT: PRINT REV $\left\{\begin{matrix};\\,\end{matrix}\right\}$ [Output]

EXPLANATION:

1. Used in a PRINT statement to display characters in reverse field.
2. REV can be canceled using the NORM function.
3. Changing modes using the CAL or MENU keys cancels the REV specification.

SEE: NORM, PRINT

SAMPLE PROGRAM:

```
10 PRINT REV; "CHARACTERS ARE REVERSED"
```

NORM

PURPOSE: Cancels the REV specification.

FORMAT: PRINT NORM $\left\{\begin{matrix};\\,\end{matrix}\right\}$ [Output]

EXAMPLE: PRINT NORM

EXPLANATION:

Used in a PRINT statement to cancel the REV specification.

SEE: REV

SAMPLE PROGRAM:

```
10 PRINT "AD 1990"
20 PRINT REV; "ABCDEF"
30 PRINT NORM; "ABCDEF"
40 FOR I=0 TO 500:NEXT I
50 END
```

Reverse field mode is specified at line 20 and continues until canceled in line 30.

PRINT USING

PURPOSE: Displays output in a specified format.

FORMAT:

```
PRINT USING "format specification"  ;  output              [{ ; }   output             ]*
            ──────────────────────     ──────────────────  [{ , }  ─────────────────── ]
            String expression          String expression            String expression
                                       or numeric expression        or numeric expression
```

EXAMPLE:

```
PRINT USING "&␣␣␣␣&" ; A$
PRINT USING "###.##" ; X
```

PARAMETERS:
1. format specification: String expression
2. output: Numeric expression or string expression

EXPLANATION:

1. Displays output in a specified format. The format can be expressed using a combination of the following characters.
   a) String formats
      !......................Only first character displayed.
      &   &.........Number of characters displayed equals the number of spaces from & to & (inclusive). When output is longer than the specified length, the string is displayed from the beginning and truncated at the specified length. When output is shorter than the length specified, the whole string is displayed, and spaces are inserted up to the specified length.
      @....................Output displayed without change.
   b) Numeric value formats
      #.....................Number of digits. Numeric values are right-justified.
      ....Decimal point position. "0" is output when the value has no fractional part.
      ,......................Used with "#" to separate integer part of value into 3-digit segments.
      ∧∧∧∧...........Used at end of expression to indicate exponent.
2. Including characters other than those noted above in the format specification causes those characters to be output as they are written.
3. Numeric data can only be formatted using numeric formats, and string data can only be formatted using string formats. A TM error is generated when an attempt is made to format string data using a numeric format, and vice versa.
4. When the value of a numeric expression is longer than the number of digits specified by the format, the number is rounded and displayed. A percent sign is displayed in front of the number when the output value is longer than the specified format.
5. Multiple formats can be written in the format specification with the formats separated by a character that does not appear in the format.
6. When the number of outputs exceeds the number of formats, the formats are applied in sequence from the first to the last. The sequence then returns to the first format.

7. A line feed is performed at the end of the display unless a semicolon is included at the end of this statement.
8. When " ∧ " is used (for exponential display), commas separating the integer into 3-digit segments are ignored, and one space is provided before the numeric value to indicate the sign of the number.

SEE: PRINT, LPRINT, LPRINT USING

SAMPLE PROGRAM:

```
10 A$=""
20 FOR I=1 TO 10
30 A$=A$+CHR$(I+64)
40 PRINT USING "&      &";A$
50 FOR T=1 TO 100:NEXT T
60 NEXT I
```

LOCATE

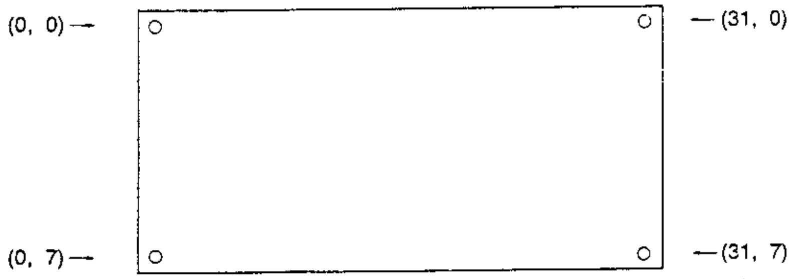
PURPOSE: Moves the cursor to a specified position on the virtual screen.

FORMAT: LOCATE $\underbrace{\text{X-coordinate,}}_{\text{Numeric expression}}$ $\underbrace{\text{Y-coordinate}}_{\text{Numeric expression}}$

EXAMPLE: LOCATE 10, 0

PARAMETERS:
1. X-coordinate: Numeric expression truncated to an integer in the range of $0 \leqq$ X-coordinate < 32
2. Y-coordinate: Numeric expression truncated to an integer in the range of $0 \leqq$ Y-coordinate < 8

EXPLANATION:
1. Locates the cursor at a specified position on the virtual screen.
2. The origin of the coordinates is the upper left corner of the screen (0, 0). The X coordinate value is incremented for each character position to the right. The Y value coordinate is incremented form each line down.



SAMPLE PROGRAM:

```
10 CLS
20 LOCATE 0,0
30 PRINT TIME$
40 GOTO 20
```

CLS

PURPOSE: Clears the display screen.

EXAMPLE: CLS

EXPLANATION:

The screen is cleared and the cursor is located at the home position. Pressing the CLS key or executing PRINT CHR$(12) ; produces the same result.

SAMPLE PROGRAM:

```
10 REM CLEAR SCREEN
20 CLS
```

DEFCHR$

PURPOSE: Defines character patterns for display.

FORMAT: DEFCHR$ $\underbrace{\text{(code)}}_{\text{Numeric expression}} = \underbrace{\text{character pattern}}_{\text{String expression}}$

EXAMPLE: DEFCHR$(240) = "04061C1C0E0E"

PARAMETERS:
1. code: Numeric expression truncated to an integer in the range of 240 ≦ code < 256 (internal character code)
2. character pattern: String expression 12 characters long, consisting of the characters 0~9 and A~F

EXPLANATION:
1. Character patterns are defined by DEFCHR$ using internal codes in the range of 240 (&HF0) ~ 255(&HFF).
2. Character patterns are defined using string expressions of hexadecimal values that represent dot patterns.
3. Display characters are formed using an 8 × 6 dot pattern, divided into 12 segments, each consisting of four dots (Fig. 1). The display status of each segment is controlled using hexadecimal values, as shown below:
   The following procedure would result in a character pattern such as the one illustrated in Fig. 2.
   a) Determine the values to be assigned to each segment:

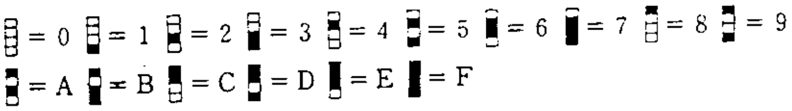


Fig. 1

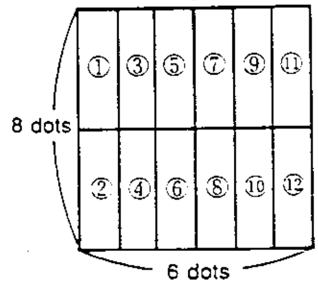
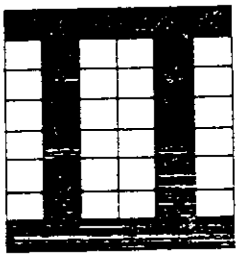


Fig. 2



   b) Arrange the values in order from pattern 1 through pattern 12.
   81FF8181FF81
   c) Assign the pattern to a character code (here to code 240, or &HF0).
   DEFCHR$ (&HF0) = "81FF8181FF81"
   d) The character pattern can be displayed using the CHR$ statement.
   PRINT CHR$(&HF0) EXE

SEE: CHARACTER CODE TABLE

SAMPLE PROGRAM:

```
10 A$=""
20 FOR I=1 TO 12
30 READ D:D$=CHR$(D):A$=A$+D$
40 NEXT I
50 DEFCHR$(240)=A$
60 PRINT CHR$(240)
70 DATA &HF,&HF,&H8,&H1,&H8,&H1
80 DATA &H8,&H1,&H8,&H1,&HF,&HF
```

Displays a frame. Changing values in lines 70 ~ 80 produces other patterns.

BEEP

PURPOSE: Sounds the buzzer and controls key input signal.

FORMAT:

$$\text{BEEP}\begin{bmatrix}0\\1\\\text{ON}\\\text{OFF}\end{bmatrix}$$

EXAMPLE: BEEP 1, BEEP ON

EXPLANATION:

1. A low tone is specified by BEEP or BEEP 0.
2. A high tone is specified by BEEP 1.
3. A tone sounds each time a key is pressed when BEEP is ON.
4. BEEP OFF switches the key tone OFF.
5. Numeric expressions can be used in place of 0 and 1.

SAMPLE PROGRAM:

```
10 BEEP 1:BEEP 0:BEEP 0:BEEP 0
```

INPUT

PURPOSE: Assigns keyboard data input to a variable.

FORMAT: INPUT ["message" $\left\{\begin{matrix};\\,\end{matrix}\right\}$] $\begin{matrix}\text{variable}\\\text{name}\end{matrix}$ [,["message" $\left\{\begin{matrix};\\,\end{matrix}\right\}$] , $\begin{matrix}\text{variable}\\\text{name}\end{matrix}$]

EXAMPLE: INPUT "YEAR=", Y, "MONTH=", M, "DAY=", D

PARAMETERS: 1. message: Character string beginning with a string constant.
2. variable name: Numeric variable name or string variable name.

EXPLANATION:

1. Data can be input to the specified variable from the keyboard.
2. Messages included in the INPUT statement are displayed. A question mark is displayed following the message when a semicolon is included following the message specification.
3. A question mark only is displayed when a message is not specified.
4. The EXE key must be pressed following each data input.
5. Numeric expressions can only be assigned to numeric variables, and string expressions can only be assigned to string variables. A TM error is generated when an attempt is made to assign a string expression to a numeric variable.
6. Quotation marks are not used when entering string data. Enclosing a string in quotation marks causes the quotation marks to be stored as part of the string.
7. Pressing the EXE key without entering data inputs a string of length 0 for a string variable, while a numeric variable retains its current value.
8. Generally, the logical line immediately following the message is input. The cursor can, however, be moved to any position on the virtual screen (using the cursor keys), and all data from the current cursor position to the end of the current logical line are input when EXE is pressed.
9. The LCKEY key, ENG key, and the touch-keys do not function during execution of the INPUT statement.
10. Numeric expressions may be used for numeric value input.
11. Pressing the ON key or changing modes during execution of the INPUT statement terminates program execution.

SAMPLE PROGRAM:

```
10 INPUT"INPUT STRING";S$
20 PRINT"S$=";S$
30 END
```

Displays string entry.

INKEY$

PURPOSE: Assigns a single character input from the keyboard to a variable.

EXAMPLE: A$ = INKEY$

EXPLANATION:

1. Returns the character or performs the function corresponding to the key pressed during execution of this statement. A null string is returned if a key is not pressed.
2. The following operations are performed when the keys listed below are pressed during execution of INKEY$.

   BRK: Terminates program execution.

   STOP: Suspends program execution.

   LCKEY, MENU, CAL, MEMO, MEMO IN, IN, OUT,
   CALC, ANS, ENG, CAPS, SHIFT + one-key commands,] Return a null string.
   F + one-key functions, CONTRAST, NEW ALL
3. The cursor is not displayed during data input stand by, and input characters are not displayed. Control codes (00H ~ IFH) can be input, but the corresponding operations will not be performed.
4. The touch-keys can be used during program execution and are read by INKEY$. The following illustration shows the character codes (&HF0 ~ &HFF) that correspond to the touch-keys.

F0	F1	F2	F3
F4	F5	F6	F7
F8	F9	FA	FB
FC	FD	FE	FF

See page 8 for more information concerning the touch-keys.

SEE: INPUT$

SAMPLE PROGRAM:

```
10 PRINT"PRESS ANY KEY"
20 C$=INKEY$
30 IF C$="" THEN 20
40 PRINT"YOU PRESS ";C$;"KEY"
50 END
```

Displays character corresponding to key input.

INPUT$

PURPOSE: Assigns a specified number of characters from the keyboard to a variable.

FORMAT: INPUT$ $\underbrace{\text{(number of characters)}}_{\text{Numeric expression}}$

EXAMPLE: A$ = INPUT$ (3)

PARAMETERS: number of characters: Numeric expression truncated to an integer in the range of $0 \leq$ number of characters < 256

EXPLANATION:

1. A string of the length specified by the number of characters is read from the keyboard buffer. Execution waits for the keyboard input when the buffer is empty.
2. The following operations are performed when the keys listed below are pressed during execution of INPUT$.
   BRK: Halts program execution.
   $\left.\begin{array}{l}\text{LCKEY, MEMO, IN, OUT, ENG, ANS}\\ \text{SHIFT + one-key commands,}\\ \text{F + one-key functions, CAPS}\end{array}\right\}$ Return a null string
3. The cursor is not displayed during data input stand by, and input characters are not displayed. Control codes (&H00 ~ &H1F) can be input, but the corresponding operations will not be performed.

SEE: INKEY$

SAMPLE PROGRAM:

```
10 PRINT "ENTER SECRET CODE"
20 ID$=INPUT$(4)
30 IF ID$<>"9876" THEN 10
40 PRINT "OK"
```

DIM

PURPOSE: Declares an array.

FORMAT:

DIM array name $\underbrace{\text{(subscript maximum value}}_{\text{Numeric expression}}$ [, $\underbrace{\text{subscript maximum value}}_{\text{Numeric expression}}$]*)

[, array name $\underbrace{\text{(subscript maximum value}}_{\text{Numeric expression}}$ [, $\underbrace{\text{subscript maximum value}}_{\text{Numeric expression}}$]*)]*

EXAMPLE: DIM A$(10), B$(10), X(2,2,2)

PARAMETERS:

1. Array name: Variable name
2. subscript maximum value: Numeric expression truncated to an integer

EXPLANATION:

1. Declares an array of the dimensions determined by the number of subscript maximum values. The size of the array is determined by each subscript maximum value.
2. Array elements range from 0 through the specified subscript maximum value.
3. All elements of a newly declared array are set to their initial value. For numeric arrays, the initial value is 0, while string arrays assigned null strings (length 0).
4. The size of an array is limited by available memory capacity. Declaration by the DIM statement is subjected to the limitations specified for logical lines (255 characters).
5. Declaring identical (same array name, same subscript maximum value) in the same program causes second declaration to be disregarded. Declaring two arrays with identical names and different subscript maximum values results in a DD error.
6. An array variable cannot be used unless they are first declared in a DIM statement.

SEE: ERASE

SAMPLE PROGRAM:

```
10 DIM A$(5)
20 FOR I=65 TO 70
30 A$(I-65)=CHR$(I)
40 PRINT A$(I-65);
50 NEXT I
```

ERASE

PURPOSE: Erases a specified array.

FORMAT: ERASE [array name [, array name]*]

EXAMPLE: ERASE A$, X

PARAMETERS: array name: Variable name

EXPLANATION:

1. Erases the specified array from memory.
2. An error does not result when the specified array does not exist, and the program proceeds to the next executable statement.
3. The ERASE statement cannot be used in a FOR－NEXT loop.
4. To declare an array using an name already assigned to an existing array, first erase the existing array with the ERASE statement.

SEE: DIM

SAMPLE PROGRAM:

```
10 CLEAR
20 DIM A$(10),B$(10)
30 ERASE A$
40 VARLIST
```

CALL

PURPOSE: Calls a machine language subroutine.

FORMAT:

$$\text{CALL}\begin{Bmatrix}\underline{\text{address}}\\ \text{Numeric expression}\\ \underline{\text{“machine language filename”}}\\ \text{String expression}\end{Bmatrix}$$

EXAMPLE: CALL 100
CALL "TEST"

PARAMETERS: 1. address: direct specification of memory (RAM) address
2. machine language filename: machine language file stored in RAM

EXPLANATION:

1. Calls a machine language subroutine stored in RAM.
2. Specifying a machine language filename moves the machine language file to the execution start address of the machine language area, followed by execution of the file. An error is generated if the size of the machine language area (reserved using the CLEAR statement) is insufficient.
3. Execution is returned to BASIC or the CAL mode by the assembler RTN command.

SEE: CLEAR, MON

PEEK

PURPOSE: Returns the value stored at the specified memory address.

FORMAT:

$$\text{PEEK}\ \underset{\text{Numeric expression}}{\underline{\text{(address)}}}$$

EXAMPLE: PEEK (&H100)

PARAMETERS: address: Numeric expression truncated to an integer in the range of $-32769 <$ address < 65536. Negative addresses are added to 65536 and the contents of the resulting address are returned (i.e. PEEK (−1) is identical to PEEK (65535)).

EXPLANATION:
Returns the value stored in memory at the specified address.

SEE: POKE

SAMPLE PROGRAM:

```
10 FOR I=&H7000 TO &H7100
20 PRINT HEX$(PEEK(I));"    ";
30 NEXT I
```

Prints memory contents from &H7000 to &H7100 in hexadecimal.

POKE

PURPOSE: Writes data to a specified address.

FORMAT: POKE address, data

	address,	data
POKE	Numeric expression	Numeric expression

EXAMPLE: POKE &H7000, 0

PARAMETERS:
1. address: Numeric expression truncated to an integer in the range of $-32769 <$ address < 65536. Negative addresses are added to 65536 and data are written to the resulting address (i.e. POKE -1, is identical to POKE 65535, data).
2. data: Numeric expression truncated to an integer in the range of $0 \leqq$ data < 256

EXPLANATION:
1. Writes data to the specified address in memory.
2. Runaway execution may result if the contents of an address outside the user work area is altered using the POKE statement.

SEE: PEEK

SAMPLE PROGRAM:

```
10 CLEAR,10
20 FOR I=&H7000 TO &H7010
30 POKE I,0
40 NEXT I
50  END
```

Clears (assigns zeros) memory from 7000H to 7010H.

DRAW
DRAWC

PURPOSE: Draws and deletes a point or a line on the screen.

FORMAT: $\left\{\begin{matrix}\text{DRAW}\\\text{DRAWC}\end{matrix}\right\}$ [−] $\underset{\text{Origin}}{\underline{(X, Y)}}$ [− $\underset{\text{Endpoint}}{\underline{(X, Y)]^{*}}}$

EXAMPLE: DRAW (50, 0) − (80, 30) − (20, 30) − (50,0)
DRAW − (50, 30)

PARAMETERS: 1. X: Numeric expression in the range $0 \leqq X < 192$
2. Y: Numeric expression in the range $0 \leqq Y < 64$

EXPLANATION:

1. Including "−" in front of (X, Y), draws a line from the last graphic pointer to the point specified by (X, Y). If "−" is not included in front of (X, Y), a point is drawn at (X, Y). Figures can be drawn by chaining multiple lines.
2. The DRAWC statement erases a line.
3. The graphic pointer is moved to the last endpoint specified.

NOTE: The graphic pointer stores the coordinates of the last point drawn using the DRAW statement.

SAMPLE PROGRAM

```
10 CLS
20 DRAW(0,16)-(191,16)
30 DRAW(0,0)-(0,31)
40 ANGLE 0
50 PRINT"SIN,COS"
60 FOR I=0 TO 191
70 DRAW(I,SIN(180+I*4)*15+16)
80 DRAW(I,COS(180+I*4)*15+16)
90 NEXT I
```

POINT

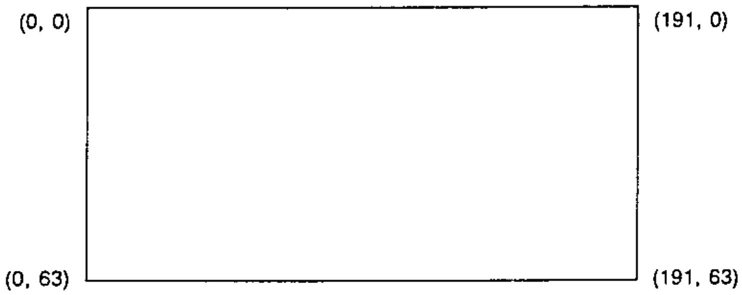
PURPOSE: Checks whether a dot on the virtual screen is lit.

FORMAT: POINT $\underbrace{\text{(X-coordinate,}}_{\text{Numeric expression}}\ \underbrace{\text{Y-coordinate)}}_{\text{Numeric expression}}$

EXAMPLE: IF POINT (10,20) - 1 THEN 30

PARAMETERS:
1. X-coordinate: Numeric expression truncated integer in the range of $0 \leqq \text{X-coordinate} < 192$
2. Y-coordinate: Numeric expression truncated integer in the range of $0 \leqq \text{Y-coordinate} < 64$

EXPLANATION:
1. A value of 1 is returned when the dot at the specified virtual screen location is lit, while value of 0 is returned when the dot is not lit.
2. The origin (0, 0) of the virtual screen is its upper left corner, while the lower right corner coordinates are (191, 63).



SAMPLE PROGRAM:

```
10 INPUT"X= TO 191":X
20 INPUT"Y= TO 13":Y
30 IF POINT(X,Y) THEN PRINT"TRUE":END
40 PRINT"FALL":END
```

"TRUE" displayed if dot is lit, and "FALL" displayed if not lit at specified coordinates.

ON ERROR GOTO

PURPOSE: Specifies the line number to which execution branches when an error is generated.

FORMAT: ON ERROR GOTO branch destination line number

Line number

EXAMPLE: ON ERROR GOTO 1000

PARAMETERS: branch destination line number:

Integer in the range of 0 ≦ line number ≦ 65535

EXPLANATION:

1. Specifies the line number to which program execution branches when an error is generated. The program returns to normal operations when a RESUME statement is executed after the error handling routine (starting at the specified line number) is executed.
2. An error is generated and program execution is halted when the branch destination line number is 0.
3. An error generated after execution branches to the specified line number causes an error message to be displayed and program execution to be halted.
4. An ON ERROR GOTO statement must be followed by a corresponding RESUME statement in the same program file. Branching to another program file using ON ERROR GOTO generates an error when the RESUME statement in the other program area is executed.
5. An OM error is generated when program execution is terminated while files are open (because of insufficient memory capacity), but error branching will not be performed.
6. The 256 characters in the I/O buffer are discarded when an OM error occurs while attempting to write to a file using the PRINT# statement.
7. Generation of an LB error while ON ERROR GOTO is specified cuts off the I/O buffer and FDD commands become inoperative.
8. The operations outlined are limited to BASIC program execution.

SEE: ERR, ERL, RESUME

SAMPLE PROGRAM:

```
10 ON ERROR GOTO 40
20 **ERROR**
30 END
40 PRINT"OOPS! ERROR!!!" : BEEP 1
50 RESUME 30
```

RESUME

PURPOSE: Returns from an error handling routine to the main routine.

FORMAT:

$$\text{RESUME}\left[\left\{\begin{array}{c}\text{NEXT}\\ \dfrac{\text{return line number}}{\text{Line number}}\end{array}\right\}\right]$$

EXAMPLE: RESUME NEXT
RESUME 100

PARAMETERS: 1. NEXT
2. return line number: Integer in the range of 1 ≦ line number ≦ 65535

EXPLANATION:

1. This statement is entered at the end of an error handling routine.
2. The statement that generated the original error is the default option when the return destination (NEXT or return line number) is omitted.
3. Program execution returns to the statement following the statement that generated the original error when NEXT is specified.
4. Return line number specifies the line to which program execution is to be resumed.
5. A RESUME statement without a return destination or a RESUME statement that specifies the line in which the original error was generated as the return line number cannot be written at the beginning of the error handling routine. This would result in an endless loop between the statement in which the error was generated and the error handling routine.
6. A RESUME statement must always be included in the same file area as the ON ERROR GOTO statement.

SEE: ERR, ERL, ON ERROR GOTO

SAMPLE PROGRAM:

```
10 ON ERROR GOTO 1000
20 INPUT A
30 D=1/A
40 PRINT "1/";A;"=";D
50 GOTO 20
1000 PRINT "0 IS ILLEGAL"
1010 RESUME 20
```

Calculates reciprocals of input values and returns to line 20 if a 0 is entered (resulting in division by 0).

ERL

PURPOSE: Returns the number of a line in which an error has been generated.

FORMAT: ER - ERL

EXPLANATION:

The value of ERL can only be changed within a program, and the value is cleared when a program is executed or when the power of the unit is switched OFF.

SEE: ERR, ON ERROR GOTO

SAMPLE PROGRAM

```
10 ON ERROR GOTO 40
20 **ERROR**
30 END
40 PRINT"ERROR LINE=";ERL
50 RESUME 30
```

Error is generated in line 20 and corresponding error code is displayed in line 40.

ERR

PURPOSE: Returns the error code which corresponds to a generated error.

FORMAT: PRINT ERR

EXPLANATION:

The value of ERR can only be changed within a program, and the value is cleared when a program is executed or when the power of the unit is switched ON. See the error message table on page 197 for details concerning error codes and their corresponding error messages.

SEE: ON ERROR GOTO, ERL, Error Message Table

SAMPLE PROGRAM:

```
10 ON ERROR GOTO 40
20 **ERROR**
30 END
40 PRINT"ERROR CODE=";ERR
50 RESUME 30
```

An error is generated in line 20 and the corresponding error code is displayed in line 40.

NUMERIC FUNCTIONS

ANGLE

PURPOSE: Specifies the angle unit.

FORMAT: ANGLE angle specification
Numeric expression

EXAMPLE: ANGLE 0

PARAMETERS: angle specification: Numeric expression truncated to an integer in the range of 0 ≦ angle specification < 3.

EXPLANATION:

1. The angle units for the trigonometric function can be specified using the values 0, 1, and 2.
   0: DEG (degrees)
   1: RAD (radians)
   2: GRAD (grads)
2. The relationships between the angle units are as follows:

Angle Unit	DEG	RAD	GRAD
1DEG =	1	$\frac{\pi}{180}$	$\frac{100}{90}$
1RAD =	$\frac{180}{\pi}$	1	$\frac{200}{\pi}$
1GRAD =	$\frac{90}{100}$	$\frac{\pi}{200}$	1

$90° = \frac{\pi}{2}$ rad = 100 grad

3. ANGLE 0 is set automatically when NEW ALL is executed.

SAMPLE PROGRAM

```
10 ANGLE 0 'DEGREE
20 PRINT SIN 30;
30 ANGLE 1 'RADIAN
40 PRINT SIN(PI/6);
50 ANGLE 2 'GRAD
60 PRINT SIN(100/3)
```

SIN
COS
TAN

PURPOSE: Returns the value of the corresponding trigonometric function value for the argument.

FORMAT:

SIN (argument)
Numeric expression

COS (argument)
Numeric expression

TAN (argument)
Numeric expression

* The parentheses enclosing the argument can be omitted when the argument is a numeric value or variable.

EXAMPLE: SIN (30), COS (PI/2)

PARAMETERS: argument: Numeric expression (angle)
|argument| < 1440 (DEG)
|argument| < 8π (RAD)
|argument| < 1600 (GRAD)

EXPLANATION:

Returns the value of the corresponding trigonometric function for the argument.

SIN	SINE
COS	COSINE
TAN	TANGENT

SEE: ANGLE, ASN, ACS, ATN

SAMPLE PROGRAM:

```
10 ANGLE 0
20 INPUT "DEGREE=",D
30 PRINT"SIN(";D;")=";SIN D
40 PRINT"COS(";D;")=";COS D
50 PRINT"TAN(";D;")=";TAN D
60 GOTO 20
```

Displays trigonometric function values for input angles.

ASN
ACS
ATN

PURPOSE: Returns the value of the corresponding inverse trigonometric function for the argument.

FORMAT:

ASN (argument)
Numeric expression

ACS (argument)
Numeric expression

ATN (argument)
Numeric expression

* The parentheses enclosing the argument can be omitted when the argument is a numeric value or variable.

EXAMPLE: ASN (0.1)

PARAMETERS: argument: Numeric expression in the range of −1 ≦ argument ≦ 1 (ASN, ACS)

EXPLANATION:

1. Returns the value of the corresponding inverse trigonometric function for the argument.
   ASN ARCSINE
   ACS ARCCOSINE
   ATN ARCTANGENT

2. Function values are returned within the following ranges:
   −90° ≦ASN (x) ≦ 90°, 0°≦ ACS (x) ≦ 180°
   −90° ≦ ATN (x) ≦ 90°

SEE: ANGLE, SIN, COS, TAN

SAMPLE PROGRAM:

```
10 ANGLE 1
20 INPUT"INPUT NUMBER(-1 TO 1)";N
30 PRINT N;"=SIN(";ASN(N);"RAD)"
40 PRINT N;"=COS(";ACS(N);"RAD)"
50 PRINT N;"=TAN(";ATN(N);"RAD)"
60 FOR I=0 TO 500:NEXT I
70 ANGLE 0:END
```

Displays trigonometric angles in radians for each input in range of −1 to 1.

HYP SIN
HYP COS
HYP TAN

PURPOSE: Returns the value of the corresponding hyperbolic function for the argument.

FORMAT:

HYP SIN (argument)
Numeric expression

HYP COS (argument)
Numeric expression

HYP TAN (argument)
Numeric expression

* The parentheses enclosing the argument can be omitted when the argument is a numeric value or variable.

EXAMPLE: HYP SIN (1.5)

PARAMETERS: argument: Numeric expression
HYP SIN |argument| ≦ 230.2585092
HYP COS |argument| ≦ 230.2585092

EXPLANATION:
Returns the value of the corresponding hyperbolic function for the argument.

HYP SIN (x) : $\sinh x = (e^x - e^{-x})/2$
HYP COS(x) : $\cosh x = (e^x + e^{-x})/2$
HYP TAN(x) : $\tanh x = (e^x - e^{-x})/(e^x + e^{-x})$

SEE: HYP ASN, HYP ACS, HYP ATN

SAMPLE PROGRAM:

```
10 INPUT"INPUT NUMBER(UP TO 230)":N
20 PRINT"HSN(";N;")=";HYPSIN N
30 PRINT"HCS(";N;")=";HYPCOS N
40 PRINT"HTN(";N;")=":HYPTAN N
50 FOR I=0 TO 500 : NEXT I
60 END
```

· Displays the hyperbolic functions for numeric input up to 230.

HYP ASN
HYP ACS
HYP ATN

PURPOSE: Returns the value of the corresponding inverse hyperbolic function for the argument.

FORMAT:

HYP ASN (argument)
Numeric expression

HYP ACS (argument)
Numeric expression

HYP ATN (argument)
Numeric expression

* The parentheses enclosing the argument can be omitted when the argument is a numeric value or variable.

EXAMPLE: HYP ASN (10)

PARAMETERS: argument: Numeric expression

HYP ASN	argument $< 5 \times 10^{99}$ (5E+99)
HYP ACS	$1 \leqq$ argument $< 5 \times 10^{99}$ (5E+99)
HYP ATN	$-1 \leqq$ argument < 1

EXPLANATION:

Returns the value of the corresponding inverse hyperbolic function for the argument.

HYP ASN(x) : $\sinh^{-1}x = \log_e(x + \sqrt{x^2+1})$

HYP ACS(x) : $\cosh^{-1}x = \log_e(x + \sqrt{x^2-1})$

HYP ATN(x) : $\tanh^{-1}x = \frac{1}{2}\log_e \frac{1+x}{1-x}$

SEE: HYP SIN, HYP COS, HYP TAN

SAMPLE PROGRAM:

```
10 INPUT"INPUT NUMBER(1 OR GREATER)";N
20 PRINT"HAS(";N;")=";HYPASN N
30 PRINT"HAC(";N;")=";HYPACS N
40 END
```

Displays inverse hyperbolic function value for numeric input of 1 or greater.

EXP

PURPOSE: Returns the value of the exponential function for the argument.

FORMAT: EXP $\underbrace{\text{(argument)}}_{\text{Numeric expression}}$

* The parentheses enclosing the argument can be omitted when the argument is a numeric value or variable.

EXAMPLE: EXP (1)

PARAMETERS: argument: Numeric expression in the range of argument ≦ 230.2585092

EXPLANATION:

Returns the value of the exponential function value for the argument.
EXP $(x) = e^x$

SEE: LOG, LGT

SAMPLE PROGRAM:

```
10 INPUT "e^X(UP TO 230)";N
20 PRINT "e^";N;"=";EXP N
30 END
```

Displays exponential function value for numeric input up to 230.

LGT
LOG

PURPOSE: Returns the value of the corresponding logarithm function for the argument.

FORMAT: LGT (argument)
Numeric expression
LOG (argument)
Numeric expression

* The parentheses enclosing the argument can be omitted when the argument is a numeric value or variable.

EXAMPLE: LGT (2), LOG (3)

PARAMETERS: argument: Numeric expression
LGT: 0 < argument
LOG: 0 < argument

EXPLANATION:
Returns the value of the corresponding logarithm function value for the argument.

LGT:	Common logarithm	$\log_{10}x$, $\log x$
LOG:	Natural logarithm	$\log_e x$, $\ln x$

SAMPLE PROGRAM:

```
10 INPUT"INPUT NUMBER":N
20 PRINT"LGT";N;"=";LGT N
30 PRINT"LOG";N;"=";LOG N
40 END
```

Displays logarithm function values for numeric input greater than 0.

SQR

PURPOSE: Returns the square root of the argument.

FORMAT: SQR $\underbrace{\text{(argument)}}_{\text{Numeric expression}}$

* The parentheses enclosing the argument can be omitted when the argument is a numeric value or variable.

EXAMPLE: SQR (4)

PARAMETERS: argument: Numeric expression in the range of 0 ≦ argument

EXPLANATION:

Returns the square root of the argument.
SQR (x) : $\sqrt{x}$

SAMPLE PROGRAM:

```
10 FOR I=0 TO 10
20 PRINT USING"SQR(##)=#.#########";I;SQR I
30 FOR J=0 TO 250 :NEXT J
40 NEXT I
50 END
```

Displays square roots of values from 0 through 10.

ABS

PURPOSE: Returns the absolute value of the argument.

FORMAT: ABS $\underbrace{\text{(argument)}}_{\text{Numeric expression}}$

* The parentheses enclosing the argument can be omitted when the argument is a numeric value or variable.

EXAMPLE: ABS (−1.5)

PARAMETERS: argument: Numeric expression

EXPLANATION:

Returns the absolute value of the argument.
ABS (x) : $|x|$

SAMPLE PROGRAM:

```
10 INPUT"INPUT NUMBERS";N
20 A=ABS N
30 PRINT N;"ABS()=";A
40 END
```

Displays the absolute value of an input value.

SGN

PURPOSE: Returns a value which corresponds to the sign of the argument.

FORMAT: SGN (argument)
Numeric expression

- The parentheses enclosing the argument can be omitted when the argument is a numeric value or variable.

EXAMPLE: SGN (A)

PARAMETERS: argument: Numeric expression

EXPLANATION:

Returns a value of −1 when the argument is negative, 0 when the argument equals 0, and 1 when the argument is positive.

Argument (X)	SGN (X)
X<0	−1
X=0	0
X>0	1

SAMPLE PROGRAM:

```
10 INPUT"INPUT NUMBER";N
20 S=SGN N
30 IF S THEN PRINT "NOT ZERO":END
40 PRINT "ZERO":END
```

Uses SGN function to determine whether or not an input value equals 0.

INT

PURPOSE: Returns the largest integer which does not exceed the value of the argument.

FORMAT: INT (argument)
Numeric expression

* The parentheses enclosing the argument can be omitted when the argument is a numeric value or variable.

EXAMPLE: INT (1.3)

PARAMETERS: argument: Numeric expression

EXPLANATION:

1. Returns the largest integer which does not exceed the value of the argument.
2. INT (*x*) is equivalent to FIX (*x*) when *x* is positive, and FIX (*x*) − 1 when *x* is negative.

SEE: FIX, FRAC

SAMPLE PROGRAM:

```
10 FOR I=1 TO 10
20 N=RND(-1)*10
30 LPRINT"INT(";N;")=";INT N
40 NEXT I
50 END
```

Converts random values to integers and outputs results to printer.

FIX

PURPOSE: Returns the integer part of the argument.

FORMAT: FIX (argument)
Numeric expression

- The parentheses enclosing the argument can be omitted when the argument is a numeric value or variable.

EXAMPLE: FIX (−1.5)

PARAMETERS: argument: Numeric expression

EXPLANATION:
Returns the integer part of the argument.

SEE: INT

SAMPLE PROGRAM:

```
10 INPUT A
20 PRINT"FIX(";A;")=";FIX A
30 GOTO 10
```

Displays the integer part of input values.

FRAC

PURPOSE: Returns the fractional part of the argument.

FORMAT: FRAC (argument)
Numeric expression

- The parentheses enclosing the argument can be omitted when the argument is a numeric value or variable.

EXAMPLE: FRAC (3.14)

PARAMETERS: argument: Numeric expression

EXPLANATION:
1. Returns the fractional part of the argument.
2. The sign (±) of the value is the same as that for the argument.

SAMPLE PROGRAM:

```
10 FOR I=1 TO 10
20 N=RND(-1)*10
30 LPRINT"FRAC(";N;")=";FRAC N
40 NEXT I
50 END
```

Isolates fractional parts of random values and outputs results to printer.

ROUND

PURPOSE: Rounds the argument at the specified digit.

FORMAT: ROUND (argument, digit)

EXAMPLE: ROUND (A, −3)

PARAMETERS: 1. argument: Numeric expression
2. digit: Numeric expression truncated to an integer in the range of −100 < digit < 100

EXPLANATION:

Rounds the argument (to the nearest whole number) at the specified digit.

SAMPLE PROGRAM:

```
10 N=RND(-1)*1000
20 PRINT N
30 INPUT "WHERE";R
40 PRINT ROUND(N,R)
50 END
```

Displays random value and then rounds value at digit specified by numeric input.
For example, responding to prompt "WHERE" with input of −2 when N = 610.5765383 produces result of 610.6.

PI

PURPOSE: Returns the value of π.

FORMAT: PI

EXAMPLE: S = 2 * PI * R

EXPLANATION:

1. Returns the value of π.
2. The value of π used for internal calculations is 3.1415926536.
3. The displayed value is rounded off to 10 digits, so the value of π is displayed as 3.141592654.

SAMPLE PROGRAM:

```
10 INPUT "RADIUS";R
20 PRINT "CIRCUMFERENCE=";2*PI*R
30 PRINT "AREA=";R^2*PI
40 FOR I=0 TO 500:NEXT I
50 END
```

Calculates circumference and area of circle after input of radius.

RND

PURPOSE: Returns a random value in the range of 0 to 1.

FORMAT: RND (argument)
Numeric expression

- The parentheses enclosing the argument can be omitted when the argument is a numeric value or variable.

EXAMPLE: RND (1) ∗ 10

PARAMETERS: argument: Numeric expression

EXPLANATION:

1. Returns a random value in the range of 0 to 1. (0 < RND (X) < 1)
2. Random numbers are generated from the same table when X > 0.
3. The last random number generated is repeated when X = 0.
4. Random numbers are generated from the random table when X < 0.
5. The same series of random numbers is generated each time a program is executed unless X < 0.

SAMPLE PROGRAM:

```
10 R=RND 1 :PRINT R
20 R=RND 0 :PRINT R
30 R=RND -1 :PRINT R
40 FOR I=1 TO 1000
50 NEXT I:GOTO 10
```

Generates random numbers using each type (positive, negative, zero) of argument.

CHARACTER FUNCTIONS

CHR$

PURPOSE: Returns a single character which corresponds to the specified character code.

FORMAT: CHR$ $\underline{\text{(code)}}$
Numeric expression

EXAMPLE: CHR$ (65)

PARAMETERS: code: Numeric expression truncated to an integer in the range of $0 \leqq \text{code} < 256$

EXPLANATION:

Variables can also be used as a parameter, and decimal parts of numeric values are truncated. A null is returned when a character does not exist for the specified character code.

SEE: ASC

SAMPLE PROGRAM:

```
10 FOR I=65 TO 90
20 PRINT CHR$(I);
30 NEXT I
```

ASC

PURPOSE: Returns the character code corresponding to the character in the first (leftmost) position of a string.

FORMAT: ASC $\underline{\text{(string)}}$
String expression

EXAMPLE: ASC ("A")

PARAMETERS: string: String expression

EXPLANATION:

1. Returns the character code corresponding to a character. The character code for the first (leftmost) character only is returned for a string of two or more characters long.
2. A value of 0 is returned for a null string.

SEE: CHR$, Character Code Table

SAMPLE PROGRAM:

```
10 INPUT"INPUT CHARACTERS";A$
20 B$=LEFT$(A$,1)
30 C=ASC(A$)
40 PRINT"FIRST CHAR=";B$;" CODE=";C
50 END
```

Displays first character and corresponding character code for string input.

STR$

PURPOSE: Converts the argument (numeric value or numeric expression value) to a string.

FORMAT: STR$ $\underbrace{\text{(argument)}}_{\text{String expression}}$

EXAMPLE: STR$ (123), STR$ (255 + 3)

PARAMETERS: argument: Numeric expression

EXPLANATION:

1. Converts decimal values specified in the argument to strings.
2. Converted positive values include a leading space and converted negative values are preceded by a minus sign.

SEE: VAL

SAMPLE PROGRAM:

```
10 INPUT"INPUT NUMBERS";N
20 S$=STR$(N)
30 C$=MID$(S$,2,1)
40 PRINT"FIRST CHARACTER=";C$
50 END
```

Converts numeric input to a string. Next, the first number of converted string is displayed as character.

VAL

PURPOSE: Converts a numeric character string to a numeric value.

FORMAT: VAL (string)
String expression

EXAMPLE: A = VAL ("345")

PARAMETERS: string: String expression

EXPLANATION:

1. Converts a numeric character string to a numeric value.
2. Numeric charactèrs are converted up to the point in the string that a non-numeric character is encountered. All subsequent characters are disregarded from the non-numeric character onwards. (i.e. when A = VAL ("123A456"), A = 123).
3. The value of this function becomes 0 when the length of the string is 0 or when the leading character is non-numeric.

SEE: STR$

SAMPLE PROGRAM:

```
10 INPUT"VALUE1 ",A$
20 INPUT"VALUE2 ",B$
30 C$=A$+B$
40 C=VAL(A$)+VAL(B$)
50 PRINT C$,C
```

Performs string addition and numeric addition of two input strings.

MID$

PURPOSE: Returns a substring of a specified length from a specified position within a string.

FORMAT: MID$ (string, position [, number of characters])

String expression Numeric expression Numeric expression

EXAMPLE: MID$ (A$, 5, 3)

PARAMETERS:

1. string: String expression
2. position: Numeric expression truncated to an integer in the range of $1 \leqq$ position < 256
3. number of characters: Numeric expression truncated to an integer in the range of $0 \leqq$ number of characters < 256. The default option is from the specified position to the end of the string when this parameter is omitted.

EXPLANATION:

1. Returns a substring of a specified length from a specified position within a string. A substring from the specified position to the end of the string is returned when the length of the substring is not specified.
2. A substring of length 0 (null) is returned when the specified position exceeds the length of the string.
3. A substring from the specified position to the end of the string is returned when the specified number of characters is greater than the number of characters from the specified position to the end of the string.

SEE: RIGHT$, LEFT$

SAMPLE PROGRAM:

```
10 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20 INPUT "1 TO 26 FROM";B
30 PRINT "1 TO";27-B;"TO";
40 INPUT E
50 S$=MID$(A$,B,E)
60 PRINT S$
70 END
```

Uses numeric input to produce alphabetic series of a specified number of characters starting from a specified location.

RIGHT$

PURPOSE: Returns a substring of a specified length counting from the right of a string.

FORMAT: RIGHT$ (string, number of characters)

RIGHT$	(string,	number of characters)
	String expression	Numeric expression

EXAMPLE: RIGHT$ ("ABCDEF", 3)

PARAMETERS:
1. string: String expression
2. number of characters: Numeric expression truncated to an integer in the range of 0 ≦ number of characters < 256.

EXPLANATION:
1. Returns a substring of a specified length counting from the right of string.
2. The entire string is returned as the substring when the specified number of characters is greater than the number of characters in the string.

SEE: MID$, LEFT$

SAMPLE PROGRAM:

```
10 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20 PRINT A$
30 INPUT"1 TO 26 HOW MANY GET";N
40 PRINT RIGHT$(A$,N)
50 END
```

Uses numeric input to display specified number of characters from end of alphabetic sequence.

LEFT$

PURPOSE: Returns a substring of a specified length counting from the left of a string.

FORMAT: LEFT$ (string, number of characters)
String expression Numeric expression

EXAMPLE: LEFT$ ("ABCDEF", 3)

PARAMETERS:
1. string: String expression
2. number of characters: Numeric expression truncated to an integer in the range of 0 ≦ number of characters < 256.

EXPLANATION:
1. Returns a substring of a specified length counting from the left of string.
2. The entire string is returned as the substring when the specified number of characters is greater than the number of characters in the string.

SEE: MID$, RIGHT$

SAMPLE PROGRAM:

```
10 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20 PRINT A$
30 INPUT"1 TO 26 HOW MANY GET";N
40 PRINT LEFT$(A$,N)
50 END
```

Uses numeric input to display specified number of characters from beginning of alphabetic sequence.

LEN

PURPOSE: Returns a value which represents the number of characters contained in a string.

FORMAT: LEN (string)
String expression

EXAMPLE: LEN (A$)

PARAMETERS: string: String expression

EXPLANATION:
Returns a value which represents the number of character contained in a string, including characters that don't appear on the display (character codes from &H0 ~ 1FH) and spaces.

SAMPLE PROGRAM

```
10 INPUT"INPUT CHARACTERS";C$
20 PRINT"LENGTH=";LEN(C$)
30 END
```

Determines the length of an input string.

HEX$

PURPOSE: Returns a hexadecimal string for a decimal value specified in the argument.

FORMAT: HEX$ (argument)
Numeric expression

EXAMPLE: HEX$ (15)

PARAMETERS: Numeric expression truncated to an integer in the range of −32769 < argument < 65536. Values more than 32767 are converted by subtracting 65536.

EXPLANATION:
Returns a 4-digit hexadecimal string for a decimal value specified in the argument.

SEE: &H

SAMPLE PROGRAM:

```
10 PRINT"DECIMAL";TAB(10);"HEX"
20 FOR I=0 TO 16
30 PRINT USING"##";I;
40 PRINT TAB(10);HEX$(I)
50 FOR J=0 TO 250 :NEXT J
60 NEXT I
70 END
```

Displays the decimal values from 0 through 16 along with their hexadecimal equivalents.

&H

PURPOSE: Converts the 1 through 4-digit hexadecimal value following &H to a decimal value.

FORMAT: &H argument
hexadecimal value

EXAMPLE: A = &HAF

PARAMETERS: $0_H \leq$ argument $\leq FFFF_H$

EXPLANATION:

1. Converts the 1 through 4-digit hexadecimal value following &H to a decimal value.
2. Hexadecimal values are formed using the values 0 through 9, plus the characters A through F.

SEE HEX$

SAMPLE PROGRAM

```
10 FOR I=&H1 TO &H10
20 PRINT HEX$(I);I
30 FOR J=0 TO 250:NEXT J
40 NEXT I
50 END
```

Displays hexadecimal values and their decimal equivalents.

DEG

PURPOSE: Converts a sexagesimal value to a decimal value.

FORMAT: $\text{DEG}\ \underbrace{\text{(degrees}}_{\text{Numeric expression}}\ \underbrace{\text{[, minutes}}_{\text{Numeric expression}}\ \underbrace{\text{[, seconds]])}}_{\text{Numeric expression}}$

EXAMPLE: DEG (1, 30, 10)

PARAMETERS: Degree, minutes, seconds: $|\text{DEG (degrees, minutes, seconds)}| < 10^{100}$

EXPLANATION:
Converts the degrees, minutes, and seconds of sexagesimal values to decimal values as follow:
DEG (degrees, minutes, seconds) = degrees + minutes/60 + seconds/3600

SAMPLE PROGRAM:

```
10 T$=TIME$
20 A=VAL(LEFT$(T$,2))
30 B=VAL(MID$(T$,3,2))
40 C=VAL(RIGHT$(T$,2))
50 PRINT T$;DEG(A,B,C)
60 END
```

Converts current time to decimal.

DMS$

PURPOSE: Converts a decimal value to a sexagesimal string.

FORMAT: $\text{DMS\$}\ \underbrace{\text{(argument)}}_{\text{Numeric expression}}$

EXAMPLE: DMS$ (1.52)

PARAMETERS: argument:
Numeric expression in the range of numeric expression $< 10^{100}$

EXPLANATION:

1. Converts decimal values to sexagesimal strings.
2. Minutes and seconds are not displayed when the argument is in the range of numeric expression $\geqq 1 \times 10^{6}$ (1E6). In this case, the absolute value of the input value is converted to a string as it is.

SAMPLE PROGRAM:

```
10 INPUT"INPUT NUMBER";N
20 PRINT"=";DMS$(N)
30 END
```

Converts input decimal values to sexagesimal strings.

TIME$

PURPOSE: Returns the current time. Also used to set the time.

FORMAT: TIME$
TIME$ - "current time string"
String

EXAMPLE: TIME$ - "08:30"

EXPLANATION:

1. TIME$ is a system variable which stores the current time. Time is expressed as an 8-character string in the format: "hh:mm:ss", where hh - hours, mm - minutes, and ss - seconds.
2. Correct time is maintained even when the power of the unit is switched OFF.
3. The following format is used to assign a string to TIME$ in order to set the time within a range of "00:00" ~ "23:59".
   TIME$ - "hh:mm"
   Leading zeros are added automatically when single digits are used for TIME$ string assignment.

SEE: DATE$

SAMPLE PROGRAM:

```
10 INPUT "hh:mm:";T$
20 TIME$=T$
30 CLS
40 LOCATE 0,0
50 PRINT TIME$
60 GOTO 40
```

Displays clock after input of current time.

DATE$

PURPOSE: Returns the current date. Also used to set the system date.

FORMAT: DATE$
DATE$ - "date string"
String

EXAMPLE: DATE$ - "05-01-1986"

EXPLANATION:

1. DATE$ is a system variable which stores the current date.
2. The current date is maintained even when the power of the unit is switched OFF.
3. The date is expressed as an 10-character string in the format: "MM-DD-YYYY", where MM - month, DD - date, and YYYY - year.
4. The following format is used to assign a string to DATE$ in order to set the date within a range of "01-01-1980" ~ "12-31-2079".
   DATE$ - "MM-DD-YYYY"
   Leading zeros are added automatically when single digits are used for setting the month and date.
5. The date assigned to DATE$ is automatically updated in accordance with the time assigned to TIME$.

SEE: TIME$

SAMPLE PROGRAM:

```
10 Y$=RIGHT$(DATE$,4)
20 M$=LEFT$(DATE$,2)
30 D$=MID$(DATE$,4,2)
40 PRINT Y$;"-";M$;"-";D$;
```

Displays current date.

STATISTIC FUNCTIONS

STAT

PURPOSE: Inputs statistical data.

FORMAT: STAT X-data [, Y-data] [; frequency]
Numeric expression Numeric expression

EXAMPLE: STAT 1, 3 ; 10

PARAMETERS:
1. X-data: Numeric expression. The previous X-data is the default value when omitted.
2. Y-data: Numeric expression. The previous Y-data is the default value when omitted.
3. Both X and Y cannot be omitted at the same time.
4. The default value for the frequency is 1.
5. Specific data can be deleted by specifying a frequency of −1 followed by the X-data and Y-data to be deleted.

SEE: STAT CLEAR

SAMPLE PROGRAM:

```
10 STAT CLEAR
20 FOR I=1 TO 10
30 X=RND(1)*10:Y=RND(1)*100
40 STAT X,Y
50 NEXT I
60 LPRINT"CNT=";CNT
70 LPRINT"COR=";COR
80 LPRINT"LRA=";LRA;"LRB=";LRB
```

STAT CLEAR

PURPOSE: Initializes the statistical memories and should always be executed immediately before performing statistical processing.

EXAMPLE: STAT CLEAR

SEE: STAT

SAMPLE PROGRAM:

```
10 STAT CLEAR
20 FOR I=1 TO 10
30 X=RND(1)*10:Y=RND(1)*100
40 STAT X,Y
50 NEXT I
60 LPRINT"SDX=";SDX;"SDY=";SDY
70 LPRINT"SDXN=";SDXN;"SDYN=";SDYN
```

CNT

PURPOSE: Returns the number of statistical data items processed.

EXAMPLE: PRINT "NUMBER OF DATA - ", CNT

EXPLANATION:

Returns the number of statistical data items input using the STAT statement.

SUMX, SUMY, SUMX2, SUMY2, SUMXY

PURPOSE: SUMX: Sum of X-data
SUMY: Sum of Y-data
SUMX2: X-data sum of squares
SUMY2: Y-data sum of squares
SUMXY: X-data and Y-data sum of products

EXAMPLE: PRINT SUMX, SUMX2, SUMY, SUMY2

EXPLANATION:

These functions respectively return cumulative totals, sums of squares and sums of products as noted below:
SUMX : Σx
SUMY : Σy
SUMX2 : Σx^2
SUMY2 : Σy^2
SUMXY : Σxy

SAMPLE PROGRAM:

```
10 STAT CLEAR
20 FOR I=1 TO 10
30 X=RND(1)*10:Y=RND(1)*100
40 STAT X,Y
50 NEXT I
60 LPRINT"SUMX=";SUMX;"SUMY=";SUMY
70 LPRINT"SUMX2=";SUMX2;"SUMY2=";SUMY2
80 LPRINT"SUMXY=";SUMXY
```

MEANX, MEANY

PURPOSE: MEANX: Returns the mean of X-data.
MEANY: Returns the mean of Y-data.

EXAMPLE: PRINT MEANX

EXPLANATION:

These functions return the means of X-data and Y-data as noted bellow:
MEANX : $\Sigma x/n$
MEANY : $\Sigma y/n$

SAMPLE PROGRAM:

```
10 STAT CLEAR
20 FOR I=1 TO 10
30 X=RND(1)*10:Y=RND(1)*100
40 STAT X,Y
50 NEXT I
60 LPRINT"MEANX=";MEANX
70 LPRINT"MEANY=";MEANY
```

SDX, SDY, SDXN, SDYN

PURPOSE: SDX: Returns the sample standard deviation of X-data.
SDY: Returns the sample standard deviation of Y-data.
SDXN: Returns the population standard deviation of X-data.
SDYN: Returns population standard deviation of Y-data.

EXAMPLE: PRINT SDX; SDY

EXPLANATION:

Return sample standard deviation and population standard deviation according to the following formulas:

$$\mathrm{SDX} : \mathrm{X}\delta_{n-1} = \sqrt{\frac{n \cdot \Sigma x^2 - (\Sigma x^2)}{n(n-1)}}$$

$$\mathrm{SDY} : \mathrm{Y}\delta_{n-1} = \sqrt{\frac{n \cdot \Sigma y^2 - (\Sigma y)^2}{n(n-1)}}$$

$$\mathrm{SDXN} : \mathrm{X}\delta_{n} = \sqrt{\frac{n \cdot \Sigma x^2 - (\Sigma x^2)}{n^2}}$$

$$\mathrm{SDYN} : \mathrm{Y}\delta_{n} = \sqrt{\frac{n \cdot \Sigma y^2 - (\Sigma y)^2}{n^2}}$$

n: number of data items

LRA, LRB

PURPOSE: LRA: Returns the linear regression constant term.
LRB: Returns the linear regression coefficient.

EXAMPLE: PRINT "a = " ; LRA, "b = " ; LRB

EXPLANATION:
Determine the linear regression constant term and linear regression coefficient.

$$\text{LRA} : \frac{\Sigma y - \text{LRB} \cdot \Sigma x}{n}$$

$$\text{LRB} : \frac{n \cdot \Sigma xy - \Sigma x \cdot \Sigma y}{n \cdot \Sigma x^2 - (\Sigma x)^2}$$

n: number of data items

COR

PURPOSE: Returns the correlation coefficient (r).

EXAMPLE: PRINT "r = " ; COR

EXPLANATION:
The value of the correlation coefficient is expressed as the following formula:

$$\text{COR} : \frac{n \cdot \Sigma xy - \Sigma x \cdot \Sigma y}{\sqrt{\{n \cdot \Sigma x^2 - (\Sigma x)^2\}\ \{n \cdot \Sigma y^2 - (\Sigma y)^2\}}}$$

n: number of data items.

EOX, EOY

PURPOSE: EOX: Returns the estimated value of X in relation to Y.
EOY: Returns the estimated value of Y in relation to X.

FORMAT: EOX argument
Numeric value

EOY argument
Numeric value

EXAMPLE: PRINT EOX1

EXPLANATION:

Return estimated values, the values of which are expressed by the following formulas:

$$\mathrm{EOX}(y) : (\hat{x}) = \frac{y - \mathrm{LRA}}{\mathrm{LRB}}$$

$$\mathrm{EOY}(x) : (\hat{y}) = \mathrm{LRA} + x \times \mathrm{LRB}$$

I/O COMMANDS

LLIST

PURPOSE: Outputs program contents to the printer.

FORMAT:

```
LLIST  ⎧ [starting line number] [– [ending line number]] ⎫
       ⎨  ‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾     ‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾    ⎬
       ⎪       Line number              Line number      ⎪
       ⎩ [.]                                             ⎭
```

EXAMPLE: LLIST 50 – 100

PARAMETERS:

1. starting line number: Program line number from which program content printout is to begin. The default option is the first line of the program.
2. ending line number: Program line number at which program content printout is to end. The default option is the last line of the program.

   Both the starting line number and ending line number are within the range of 1 ≦ line number ≦ 65535. The last line number used by BASIC is specified when "." is used.

EXPLANATION:

1. Outputs program contents to the printer within the specified range.
2. This statement differs from LIST in that output is to the printer without showing program contents on the display.
3. LLIST cannot be used in the CAL mode.

SEE: LIST

LPRINT

PURPOSE: Outputs text to the printer.

FORMAT: LPRINT [output data] [$\left\{ \begin{matrix} , \\ ; \end{matrix} \right\}$ [output data]]*

EXAMPLE: LPRINT A, B

PARAMETERS: output data: Output control function, numeric expression, or string expression

EXPLANATION:

1. Outputs data to the printer. When the output data is a control function, the corresponding operation is performed. Numeric or string expressions as output data result in printout of the resulting value.
2. Numeric expression values are printed in decimal, and the print format is the same as that for the PRINT statement (see PRINT).
3. String expression values are output as they are to the printer.
4. Including a comma between output data causes a zone tab to be inserted between output data at output.
   Zone tabs are set at 14-character intervals (counting from 0, within a range of 255 characters) following the last carrier return instruction, and zone tab outputs spaces from the current location to the next zone tab. Consequently, the printing of the first character of an output data following a comma is performed at the next zone tab.

```
10 LPRINT
20 FOR I=1 TO 20:LPRINT"*",:NEXT I
30 LPRINT
40 END
```

5. Including a semicolon between output data causes the output data to be output sequentially.

```
10 LPRINT
20 FOR I=1 TO 50
30 LPRINT "(";I;")"
40 NEXT I
50 LPRINT
60 END
```

6. Including a semicolon at the end of the statement causes the location immediately following printout of the last output data to be the next printing position.
7. Including a comma at the end of an LPRINT statement performs a zone tab following printout of the last output data.

8. A carrier return is performed when a semicolon or comma is not included at the end of the statement. Print positions are counted from 0 through 255, and the count is reset to 0 when it exceeds 255. Zone tabs and the TAB function are performed in accordance with the print position count. CR-LF (internal code 0DH, 0AH) is performed at this time.
9. Actual printing begins when a carrier return/line feed code is sent, and carrier return/line feed is performed automatically when printing reaches the extreme right of the paper.

SEE: PRINT

SAMPLE PROGRAM:

```
10 LPRINT
20 FOR I=1 TO 14:LPRINT"*";I,:NEXT I
30 LPRINT
40 END
```

LPRINT USING

PURPOSE: Outputs data to the printer according to a specified format.

FORMAT:

LPRINT USING	format specification;	output data	[{ , ; } output data]*
	String expression	String or numeric expression	String or numeric expression

EXAMPLE: LPRINT USING "# # # #" ; A ; B

PARAMETERS: 1. format specification: String one or more character long
2. output data: numeric or string expression

EXPLANATION:
Outputs data to the printer according to a specified format. See the PRINT USING statement for details on the formats available.

SEE: PRINT USING

SAMPLE PROGRAM:

```
10 ANGLE 0
20 LPRINT"USING";TAB(8);"PRINT"
30 FOR I=0 TO 90 STEP 10
40 LPRINT USING"#.#####";SIN(I);
50 LPRINT SIN(I)
60 NEXT I
```

OPEN

PURPOSE: Declares a file open for use.

FORMAT:

$$\text{OPEN "file descriptor"} \left[\text{FOR}\left\{\begin{array}{l}\text{INPUT}\\ \text{OUTPUT}\\ \text{APPEND}\end{array}\right\}\right] \text{AS[\#]}\ \underbrace{\underline{\text{file number}}}_{\text{Numeric expression}}$$

EXAMPLE: OPEN "DATA1" FOR OUTPUT AS #1

PARAMETERS:
1. file descriptor: String expression
2. file number: Numeric expression truncated to an integer in the range of $1 \leq$ file number < 16

EXPLANATION:

1. Opens the file specified by the file descriptor as the specified file number. Subsequent input to and output from open files is performed by designating the file numbers.
2. Internal memory is the default option when the device name is omitted from the file descriptor.
3. Specifying FOR INPUT makes sequential file input possible. An error is generated when the specified file does not exist in internal memory and on the disk (when a floppy disk drive is being used).
4. Specifying FOR OUTPUT makes sequential file output possible, and a new file is created in internal memory or on the disk. Any currently existing file with the same filename is deleted at this time.
5. Specifying FOR APPEND makes output possible to a currently existing sequential file, and locates the file buffer pointer at the end of the file. An NF error is generated when the specified file does not exist.
6. The following two conditions are specified when either FOR INPUT, FOR OUTPUT, or FOR APPEND are not specified:
   a) FDD (0:)
      Random file access when a previously created file exists on the disk. If a file does not already exist, a new file is created.
   b) Communication circuit (COM0:)
      Sequential access, but the file can be opened as a random file.
   * Random file access cannot be specified for devices other than the FDD or communication circuit (i.e. internal memory, cassette tape recorder).
7. Sequential files only can be opened with internal memory.
8. APPEND OPEN cannot be specified for BASIC files, machine language files or random files (including files stored on floppy disk).
9. The size of a single record is automatically set at 256 bytes for random file access.
10. An OP error is generated when an attempt is made to open a file which has already been opened.
11. The file buffer is automatically retained, but an OM error is generated when the retained I/O buffer area becomes full because of execution of the CLEAR statement.
12. The OPEN statement can only be executed within a program.
13. Do not change disks while a file is opened.

SEE: CLOSE, FIELD

SAMPLE PROGRAM:

```
10 OPEN"TEST" FOR OUTPUT AS #1
20 PRINT #1,"WRITE TEST"
30 CLOSE #1
40 OPEN"TEST" FOR INPUT AS #1
50 INPUT #1,CH$
60 CLOSE #1
70 PRINT "READ->";CH$
```

Creates sequential file in internal memory.

CLOSE

PURPOSE: Closes files and declares an end to the use of the I/O (input/output) buffer.

FORMAT: CLOSE [[#] file number] [, [#] file number]*

Numeric expression Numeric expression

EXAMPLE: CLOSE #1

PARAMETERS: file number: Numeric expression truncated to an integer in the range of 1 ≦ file number < 16

EXPLANATION:

1. Closes the file specified by the file number.
2. All presently opened files are the default option when the file number is omitted.
3. An error is not generated when an attempt is made to close a file that has not been previously opened.
4. This statement also clears the I/O buffer.
5. All opened files are automatically closed at the end of a program even if the CLOSE statement is not executed.
6. An OM error is generated for files which cannot be closed because of insufficient memory capacity.

SEE: OPEN

SAMPLE PROGRAM:

```
10 OPEN"0:TEST" FOR INPUT AS #1
20 INPUT #1,A$:PRINT A$;
30 IF EOF(1)=0 THEN 20
40 CLOSE #1
```

PRINT#

PURPOSE: Outputs data to a sequential file.

FORMAT: PRINT# $\underset{\text{Numeric expression}}{\underline{\text{file number}}}$ [, output data [$\left\{\begin{matrix} , \\ ; \end{matrix}\right\}$ [output data]]*]

Output data: $\left[\begin{matrix}\text{TAB ( )} \\ \text{String expression} \\ \text{Numeric expression}\end{matrix}\right]$

EXAMPLE: PRINT #1, A$

PARAMETERS: file number: Numeric expression truncated to an integer in the range of 1 ≦ file number < 16

EXPLANATION:

1. Sequentially outputs data to the sequential file specified by the file number.
2. The contents of the output data are the same as those output to the printer by the LPRINT statement (see LPRINT, PRINT).
3. A CR-LF (0DH, 0AH) is output following the last output data when a semicolon and comma are not included.
4. This statement is only valid for sequential output, and for communication circuit (COM0:) input/output files (see OPEN).
5. Multiple data items can be output with a single PRINT# execution using commas as follows:
   PRINT #1, A ; " , " ; B ; " , " ; C$

SEE: PRINT# USING, INPUT#, PRINT, LPRINT

SAMPLE PROGRAM:

```
10 OPEN"0:TEST" FOR OUTPUT AS #1
20 INPUT"DATA=",A$
30 IF A$="" THEN 60
40 PRINT #1,A$
50 GOTO 20
60 CLOSE:END
```

PRINT # USING

PURPOSE: Outputs data to a sequential file according to a specified format.

FORMAT: PRINT# file number, USING
Numeric expression
format specification ; output data [{ ; , } output data] *
String expression   Numeric or string expression   Numeric or string expression

EXAMPLE: PRINT #1, USING "& &" ; A$

PARAMETERS:
1. file number: Numeric expression truncated to an integer in the range of 1 ≦ file number < 16
2. format specification: String expression
3. output data: Numeric or string expression

EXPLANATION:
1. Outputs data to the file specified by the file number according to the specified format.
2. Format specifications are the same as those used with the PRINT USING statement (see PRINT USING).
3. This statement is only valid for sequential output, and for communication circuit (COM0:) input/output files (see OPEN).

SEE: PRINT USING, PRINT#, OPEN

SAMPLE PROGRAM:

```
10 OPEN"TEST" FOR OUTPUT AS #1
20 PRINT #1,USING"#.##";PI
30 PRINT #1,PI
40 CLOSE #1
50 OPEN"TEST" FOR INPUT AS #1
60 IF EOF(1)=-1 THEN CLOSE #1:END
70 INPUT #1,A
80 PRINT A;
90 GOTO 60
```

INPUT#

PURPOSE: Reads data from a sequential file.

FORMAT: INPUT# file number, variable name [, variable name]*
Numeric expression

EXAMPLE: INPUT #1, A

PARAMETERS: file number: Numeric expression truncated to an integer in the range of 1 ≦ file number < 16

EXPLANATION:

1. Reads data from the file specified by the file number.
2. Data are input in the same format as data input using the INPUT statement (see INPUT). Consequently, data are delimited using commas, quotation marks, CR codes (0DH) or CR, LF codes (0DH, 0AH). Internal codes 00H through 1FH and 7FH cannot be input, and leading spaces (spaces preceding that data) are disregarded.
3. This statement is only valid for sequential input, and for communication circuit (COM0:) input/output files (see OPEN).
4. Spaces can also be used as delimiters when data are read to numeric variables.

SEE: LINE INPUT#

SAMPLE PROGRAM:

```
10 OPEN"0:TEST" FOR INPUT AS #2
20 INPUT #2,A$
30 PRINT A$;
40 IF EOF(2)=0 THEN 20
50 CLOSE #2:END
```

LINE INPUT#

PURPOSE: Inputs a single line of data from a sequential file.

FORMAT: LINE INPUT# file number, string variable name
Numeric expression

EXAMPLE: LINE INPUT#1 , A$

PARAMETERS: 1. file number: Numeric expression truncated to an integer in the range of 1 ≦ file number < 16
2. string variable name

EXPLANATION:

1. Inputs one line only from the file specified by the file number.
2. Control codes (00H ~ 1FH) and 7FH are not read.
3. Reading of data continues until a CR code (0DH) or CR, LF code (0DH, 0AH) is encountered.

SEE: INPUT#, INPUT$

SAMPLE PROGRAM:

```
10 A$="ABCDE" :B$="AD1990"
20 OPEN"TEST" FOR OUTPUT AS #1
30 PRINT #1,A$,B$
40 CLOSE #1
50 OPEN"TEST" FOR INPUT AS #1
60 LINE INPUT #1,CH$
70 CLOSE #1
80 PRINT CH$
```

INPUT$

PURPOSE: Reads the specified number of characters from a sequential file.

FORMAT: INPUT$ $\underbrace{\text{(number of characters,}}_{\text{Numeric expression}}$ [#] $\underbrace{\text{file number)}}_{\text{Numeric expression}}$

EXAMPLE: INPUT$ (16, #1)

PARAMETERS:
1. number of characters: Numeric expression truncated to an integer in the range of 0 ≦ number of characters < 256
2. file number: Numeric expression truncated to an integer in the range of 1 ≦ file number < 16

EXPLANATION:
1. Reads the specified number of characters from a sequential file.
2. All codes (00H ~ FFH) are read as they are.
3. This statement is only valid for sequential input, and for communication circuit (COM0:) input/output files (see OPEN).

SAMPLE PROGRAM:

```
10 OPEN"TEST" FOR OUTPUT AS #1
20 PRINT #1,"ABCDE AD1990"
30 CLOSE #1
40 OPEN"TEST" FOR INPUT AS #1
50 CH$=INPUT$(5,#1)
60 CLOSE #1
70 PRINT CH$
```

EOF

PURPOSE: Indicates the end of file reading.

FORMAT: EOF $\underbrace{\text{(file number)}}_{\text{Numeric expression}}$

EXAMPLE: IF EOF (1) THEN END

PARAMETERS: file number: Numeric expression truncated to an integer in the range of 1 ≦ file number < 16

EXPLANATION:
1. Indicates the end of reading for the file specified by the file number. Generally, this function is assigned a value of 0, but the value becomes −1 when the last record of a file is read.
2. A value of −1 is returned when the receive buffer (for RS-232C applications) becomes empty.
3. This statement is only valid for sequential input (see OPEN).

FIELD

PURPOSE: Allocates string variables to an I/O (input/output) buffer.

FORMAT:

```
FIELD [#] file number  [, @], string length  AS  string variable name
          ___________         _____________
          Numeric             Numeric
          expression          expression
                        [ , string length  AS  string variable name]*
                            _____________
                            Numeric
                            expression
```

EXAMPLE: FIELD #1, 40 AS N$, 20 AS T$

PARAMETERS:
1. file number: Numeric expression truncated to an integer in the range of 1 ≦ file number < 16
2. string length: Numeric expression truncated to an integer in the range of 0 ≦ string length < 256
3. string variable name: Array variables cannot be used.

EXPLANATION:
1. Allocates string variables to an I/O buffer which remain valid until the CLOSE statement is executed.
2. This statement is executed before GET and PUT statements to allow reading data from (GET) or writing data to (PUT) a random file.
3. The total of the specified string lengths cannot exceed 256 characters.
4. The FIELD statement can be executed any number of times for a single I/O buffer for assignment of data to the specified variables. Different FIELD statements can be used for each GET and PUT.
5. Using variables that appear in a FIELD statement as variables to the left of the equals sign in an INPUT or LET statement before execution of CLOSE clears the FIELD flag and converts the variables to normal (non-FIELD) string variables. The LSET or RSET statement should always be used for assignment of data to a FIELD variable.
6. Variables cannot be allocated using a FIELD statement until a file is opened using the OPEN statement.
7. Following the file number with " @ " causes this statement to be regarded as a multiple FIELD statement which is used when variable allocation cannot be performed by a single FIELD variable.
8. Executing the CLOSE statement assigns null data (three characters long) to the FIELD function.

SEE: OPEN, GET, PUT, LSET, RSET

RSET

PURPOSE: Transfers the contents of a string expression to the random I/O (input/output) buffer.

FORMAT: RSET string variable name = string expression

EXAMPLE: RSET T$ = "TEL DATA"

PARAMETERS: 1. string variable name
2. string expression

EXPLANATION:

1. Sets the contents of a string expression as right flush in the area of a string variable allocated by the FIELD statement.
2. Only string variables allocated to the I/O buffer by the FIELD statement can be used.
3. String variables defined by appearing to the left of the equals sign in INPUT or LET statement cannot be used.
4. Excess data are truncated when the string expression length is greater than the length determined by the FIELD statement.
5. Spaces (20H) are inserted when the string expression length is less than the length determined by the FIELD statement.
6. This statement differs from assignment statements in that the I/O buffer allocation is not altered after assignment.

SEE: FIELD, PUT, LSET

SAMPLE PROGRAM:

```
10 OPEN"0:TEST" AS #1
20 FIELD #1,10 AS S$
30 LSET S$="ABCDE"
40 PUT #1,1
50 RSET S$="AD1990"
60 PUT #1,2
70 CLOSE #1
80 END
```

Creates a random access file named "TEST" and writes the data "ABCDE" and "AD1990" to the file.

LSET

PURPOSE: Transfers the contents of a string expression to the random I/O (input/output) buffer.

FORMAT: LSET string variable name = string expression

EXAMPLE: LSET N$ = "NAME DATA"

PARAMETERS:
1. string variable name
2. string expression

EXPLANATION:
1. Sets the contents of a string expression as left flush in the area of a string variable allocated by the FIELD statement.
2. Only string variables allocated to the I/O buffer by the FIELD statement can be used.
3. String variables defined by appearing to the left of the equals sign in INPUT or LET statement cannot be used.
4. Excess data are truncated when the string expression length is greater than the length determined by the FIELD statement.
5. Spaces (20H) are inserted when the string expression length is less than the length determined by the FIELD statement.
6. This statement differs from assignment statements in that the I/O buffer allocation is not altered after assignment.

SEE: FIELD, PUT, RSET

PUT

PURPOSE: Writes I/O (input/output) buffer data to a file.

FORMAT: $\text{PUT}\ \underset{\text{Numeric expression}}{\underline{\text{ file number, }}}\ \underset{\text{Numeric expression}}{\underline{\text{ record number }}}$

EXAMPLE: PUT #1, X

PARAMETERS:
1. file number: Numeric expression truncated to an integer in the range of $1 \leqq$ file number < 16
2. record number: Numeric expression truncated to an integer in the range of $1 \leqq$ file record number < 1264

EXPLANATION:
1. Writes the contents of the I/O buffer to the file specified by the file number at the record specified by the record number.
2. This statement is only valid for random access files input (see OPEN).
3. This statement is only valid for floppy disk files, and is not valid for RS-232C random files.

SEE: OPEN, FIELD, GET, LSET, RSET

GET

PURPOSE: Reads data from a disk file to the I/O (input output) buffer.

FORMAT: $\text{GET [\#]}\ \underset{\text{Numeric expression}}{\underline{\text{ file number, }}}\ \underset{\text{Numeric expression}}{\underline{\text{ record number }}}$

EXAMPLE: GET #1, X

PARAMETERS:
1. file number: Numeric expression truncated to an integer in the range of $1 \leqq$ file number < 16
2. record number: Numeric expression truncated to an integer in the range of $1 \leqq$ file record number < 1264

EXPLANATION:
1. Reads to the I/O buffer the contents of the record specified by the record number from the file specified by the file number.
2. This statement is only valid for files opened as random access files (see OPEN).
3. This statement is only valid for floppy disk files, and is not valid for RS-232C random files.

SEE: OPEN, FIELD, PUT, LSET, RSET

LOF

PURPOSE: Returns the number of record during random access. During RS-232C operations, returns the remaining number of bytes in the RS-232C receive buffer.

FORMAT: LOF $\underbrace{\text{(file number)}}_{\text{Numeric expression}}$

EXAMPLE: RE - LOF (1)

PARAMETERS: file number: Numeric expression truncated to an integer in the range of 1 ≦ file number < 16

EXPLANATION:

1. An error is generated when the file number of an unopened file is specified.
2. The maximum values of LOF are as noted below:

Device	LOF
Floppy disk	1263
RS-232C	255

3. This statement is only valid with random access files.

SAMPLE PROGRAM:

```
10 OPEN"0:TEST" AS #1
20 FIELD #1,10 AS S$
30 F=LOF(1)
40 FOR I=1 TO F
50 GET #1,I
60 PRINT S$
70 NEXT I
80 CLOSE #1
90 END
```

Displays the contents of random access file "TEST".

FORMAT

PURPOSE: Initializes a floppy disk.

FORMAT: FORMAT

EXPLANATION:

1. Initializes a floppy disk.
2. Note that all of the contents written on a floppy disk are erased when it is formatted.
3. An OP error is generated when an attempt is made to format a floppy disk while a disk drive file is opened.
4. New floppy disks must always be formatted before they can be used for data storage.
5. An NR error is generated when an attempt is made to execute this command when the floppy disk drive unit is not connected.

SAMPLE PROGRAM:

```
10 PRINT"THIS IS FORMAT PROGRAM"
20 PRINT"INSERT A DISK AND PUSH ANY KEY"
30 A$=INPUT$(1)
40 FORMAT
50 PRINT"FORMAT COMPLETED"
60 END
```

Formats a new floppy disk.

BSAVE

PURPOSE: Stores memory contents directly into a specified file.

FORMAT:

```
BSAVE "file descriptor", starting address, length, [execution start address]
       String expression  Numeric          Numeric    Numeric
                          expression       expression expression
```

EXAMPLE: BSAVE "0:TEST", &H7000, &H100, &H7000

PARAMETERS:

1. file descriptor: String expression
2. beginning address: Specifies the save starting address as an offset in the range of 0000H ~ FFFFH.
3. length: Specifies the memory length (size) required for the save operation in the range of 0001H ~ FFFFH.
4. Execution start address: Specifies the execution start address of the machine language program in the range of 6FFAH ~ 7FFEH.

EXPLANATION:

1. Directly saves memory contents to a file.
2. Internal memory is the default option when the device name is omitted from the file descriptor.
3. An error is generated and save is not performed when file descriptor specification is improper.
4. An error is generated when erroneous specifications are made for the starting address or length.
5. A machine language program must be located within the address range of 6FFAH ~ 7FFEH for execution.
6. An AM error is generated and execution is not performed when CALL or MENU is used for execution and an execution start address is not input using BSAVE.
7. Files saved to internal memory using BSAVE cannot have the same filenames as previously stored BASIC or sequential files.
8. Specifying an execution start address outside the range of 6FFAH ~ 7FFEH may result in runaway execution.

SEE: BLOAD, CALL

SAMPLE EXECUTION:

```
BSAVE "0 : TEST", &H7000, &H100, &H7000
```

Save contents from 7000H to 7100H (length 100H) to floppy disk under filename "TEST".

BLOAD

PURPOSE: Loads a file to internal memory.

FORMAT: BLOAD $\underbrace{\text{"file descriptor"}}_{\text{String expression}}$ [, $\underbrace{\text{[load start address]}}_{\text{Numeric expression}}$ [, R]]

EXAMPLE: BLOAD "0:TEST", &H7000, R

PARAMETERS:

1. file descriptor: String expression
2. load start address: Specifies the load start address as an offset in the range of 0000H ~ FFFFH.
3. R: Causes immediate execution of a machine language program once it is loaded. This specification is limited to machine language programs within the address range of 7000H ~ 7FFEH with the execution start address specified when saved using BSAVE.

EXPLANATION:

1. Loads the file (i.e. machine language program) specified by the file descriptor to internal memory, starting from the specified load start address.
2. Internal memory is the default option when the device name is omitted from the file descriptor.
3. An error is generated and load is not performed when file descriptor specification is improper.
4. Loading is performed from the address specified by the BSAVE command when the load start address is omitted.
5. The "R" option can be specified to execute a machine language program immediately after loading is complete. The use of this option is limited, however, to programs saved using the BSAVE command in the address range of 7000H ~ 7FFEH. Also, execution cannot be performed unless an execution start address was specified.
6. The BLOAD command does not perform memory area check, so load can be performed anywhere in memory. Consequently, care must be exercised to avoid loading within important files, in BASIC variable areas, or within BASIC programs already stored in memory.

SEE: CLEAR, BSAVE, CALL

SAMPLE EXECUTION: BLOAD "0 : TEST", &H7000, R

Loads the file "TEST" from a floppy disk to internal memory beginning from address 7000H and executes program.

SAVE

PURPOSE: Saves a program to a specified file.

FORMAT: SAVE "file descriptor" [, A]
String expression

EXAMPLE: SAVE "DEMO1"

PARAMETERS:
1. file descriptor: String expression
2. A: Specifies ASCII format. Internal format is the default option when omitted.

EXPLANATION:
1. Outputs the currently specified program contents to the file specified by the file descriptor.
2. Internal memory is the default option when the device name is omitted from the file descriptor.
3. Programs are output in internal format (binary) when the "A" specification is omitted.
4. Specifying "A" causes the program to be converted to and saved in ASCII format which uses alphabetic characters such as those which appear when the LIST command is executed.
5. This command closes all open files and the computer waits for command input once save is complete.
6. Programs are saved in ASCII format regardless of the "A" specification when COM0: is specified for the file descriptor.
7. Programs for which a password has been specified cannot be saved using ASCII format.

SEE: LOAD, PASS, MERGE, CHAIN

SAMPLE EXECUTION: SAVE "0 : TEST"

Saves a program to a floppy disk under the filename "TEST".

LOAD

PURPOSE: Reads from a file into memory.

FORMAT: LOAD "file descriptor"
String expression

EXAMPLE: LOAD "DEMO1"

PARAMETERS: file descriptor: String expression

EXPLANATION:

1. Reads from the file specified by the file descriptor to the currently specified program area. The format of the file can be either internal or ASCII format.
2. Internal memory is the default option when the device name is omitted from the file descriptor.
3. Any program already present in the present program area is erased when this command is executed.
4. This command closes all open files and the computer waits for command input once load is complete.
5. LOAD file password current file password.

LOAD file password Current file password	"A"	"B"	None
"A"	○	×	○
"B"	×	○	○
None	○	○	○

Circled programs can be loaded.

SEE: SAVE, PASS, CHAIN

SAMPLE EXECUTION: LOAD "0 : TEST"

Reads the file "TEST" from a floppy disk file.

CHAIN

PURPOSE: Calls and executes the program in the file specified by the file descriptor.

FORMAT: CHAIN "file descriptor"
String expression

EXAMPLE: CHAIN "DEMO3"

PARAMETERS: file descriptor: String expression

EXPLANATION:

1. Calls and executes a program from the file specified by the file descriptor, and erases any program currently contained in memory.
2. Internal memory is the default option when the device name is omitted from the file descriptor.
3. Chaining a program to which a password is assigned to a program without a password results in a program with the password of the chained program.
4. Only programs without passwords assigned or a program with the same password as that assigned to the current file can be chained when a password has been specified for the current file.

CHAIN file password Current file password	"A"	"B"	None
"A"	○	×	○
"B"	×	○	○
None	○	○	○

Circled programs can be chained.

5. This command closes all open files.

SAMPLE PROGRAM:

```
10 CLS
20 PRINT"NOW LOADING"
30 CHAIN"TEST"
40 END
```

Calls and executes the file "TEST" located in internal memory.

MERGE

PURPOSE: Merges the program in the file specified by the file descriptor with a program currently stored in memory.

FORMAT: MERGE "file descriptor"
Numeric expression

EXAMPLE: MERGE "DEMO2"

PARAMETERS: file descriptor: String expression

EXPLANATION:

1. Appends the program specified by the file descriptor to the program currently present in memory.
   a) All of the program lines included in the current program are merged with all of the program lines included in the file program to create a new program as long as the line numbers for both the merged programs are different.
   b) When the two merged programs contain program lines with identical line numbers, the file program lines take priority and are included in the new program.
2. Internal memory is the default option when the device name is omitted from the file descriptor.
3. The file program must be in ASCII format.
4. This command closes all open files and the computer waits for command input once merge is complete.
5. Only programs without passwords assigned or a program with the same password as that assigned to the current file can be merged when a password has been specified for the current file.

SAMPLE EXECUTION: MERGE "0 : TEST"

Merges the disk file "TEST" with a program currently stored in memory.

VERIFY

PURPOSE: Verifies the contents of a file stored on cassette tape.

FORMAT: VERIFY "CAS0: filename"

EXAMPLE: VERIFY "CAS0: DEMO"

PARAMETERS: file descriptor: String expression

EXPLANATION:

1. Verifies the contents of a file stored on cassette tape.
2. Parity and checksum data included within the file itself are used for checking.
3. This command can be executed either in the CAL mode or BASIC mode.
4. This command closes all open files.

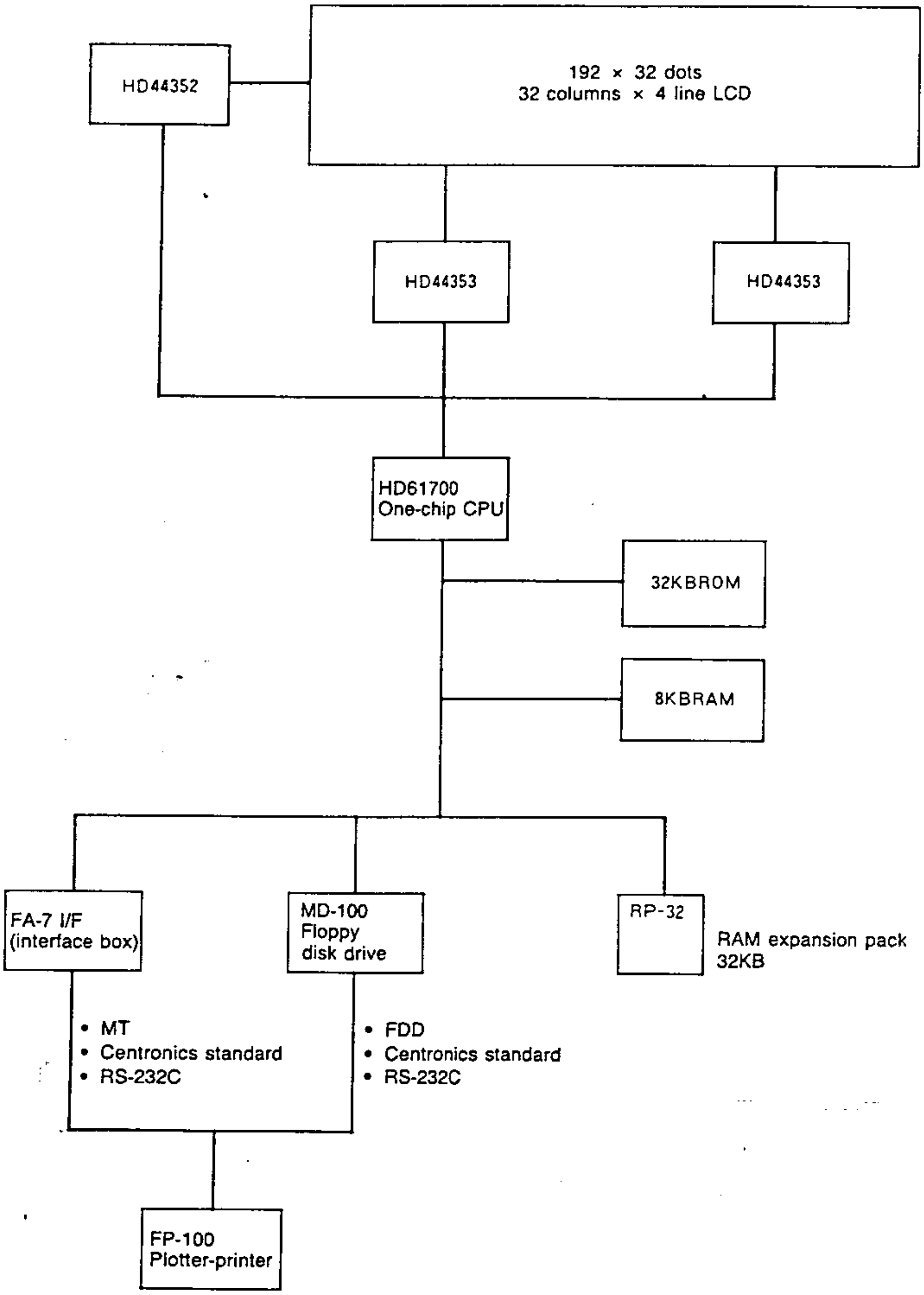
SEE: SAVE, LOAD, BSAVE, BLOAD

SAMPLE EXECUTION: VERIFY "CAS0 : TEST"

PART 2
ASSEMBLER REFERENCE

2-1 HARDWARE CONFIGURATION

This unit is composed of a 32K byte ROM and 8K byte RAM. Actual processing is performed by a custom HD61700 LSI, which is a one-chip processor with 32 bytes of RAM and 3072 words of ROM built-in. System upgrading is made possible by an optional RAM expansion pack and a selection of peripheral devices.



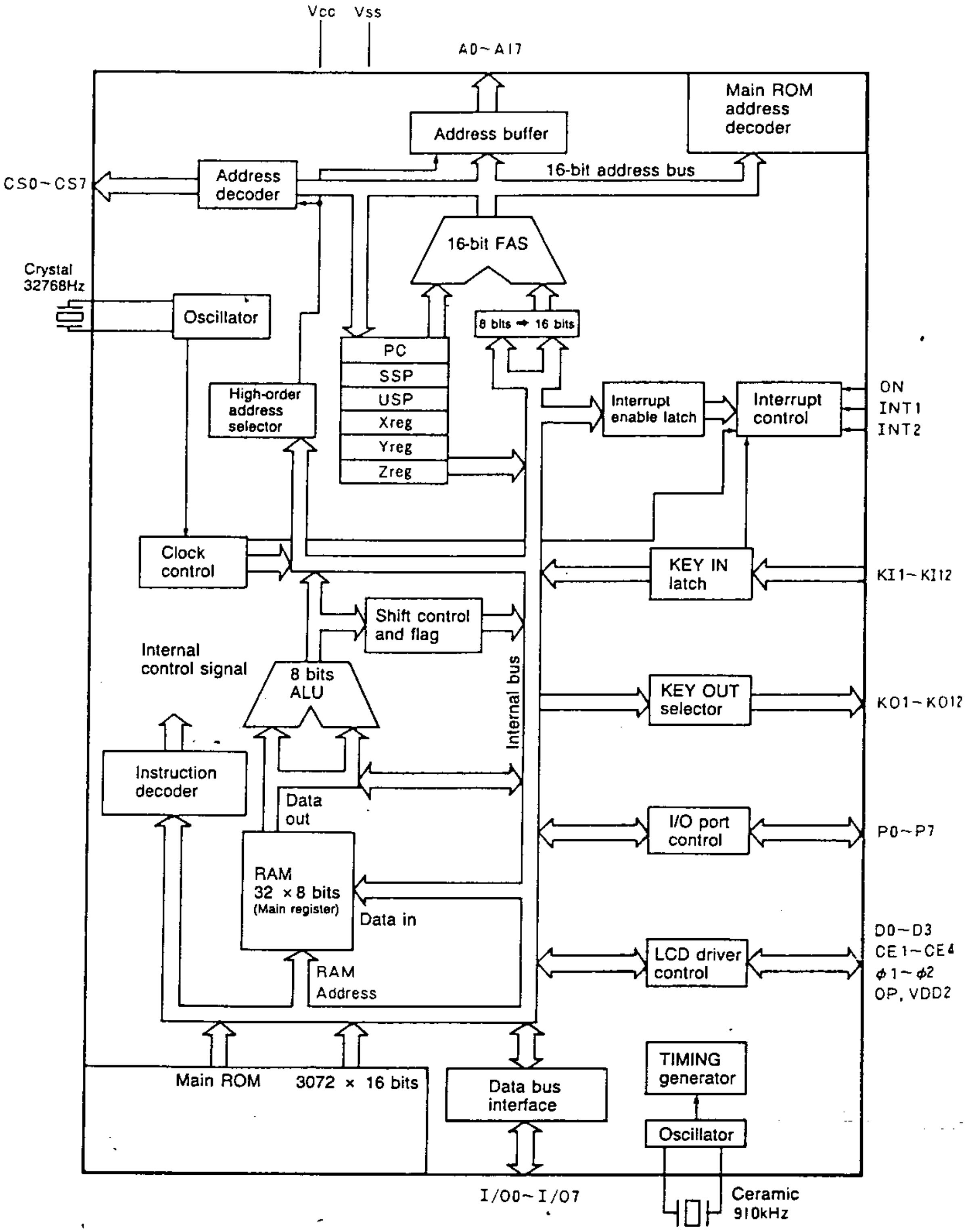
2-2 HD61700 OUTLINE

Features

The CMOS static configuration of the HD61700 results in an 8-bit microprocessor with built-in ROM and RAM. The HD61700 CPU has the following features:

- Decimal system calculation handling
- Direct access to 256K bytes of memory area
- Built-in 16-bit ROM (3072 words) for high speed processing
- Low power consumption of 800μA (maximum)
- Built-in 32 × 8 bit ROM for access in word units
- Built-in clock function (crystal with 32,768Hz)
- Key terminals 12 × 11 + 1
- Interrupt function
  Three input terminals
  KEY/Pulse
  One-minute timer (With power ON function)
- 8-bit input/output ports (software switching for input/output)
- Display control function

HD61700 Block Diagram



Internal Registers

The 32 8-bit registers and six 16-bit registers are all of CMOS static RAM configuration.

1. Main Register

The main registers are shown in the HD61700 Block Diagram marked as "RAM (32 × 8 bits)". Addresses 0 through 31 are identical to general RAM addresses, and 16-bit data can be handled using any two main registers in combination.

2. Other Registers

- **Program Counter (PC) 16-bit**

The program counter indicates the final address of the current execution. A value of one is added when execution is complete, and the next command is fetched. Addresses newly specified by jump or call commands are also set in this register.
In the case of the RETURN command, the address popped from the stack is set in the program counter.

- **Stack pointers (SSP, USP) 16-bit**

The HD61700 has two stack pointers, a system stack pointer and a user stack pointer.
Just as with a general stack pointer, the system stack pointer (SSP) saves the present program counter address to the stack when CALL commands or interrupt handling routines are encountered. This address is restored to the program counter upon return or interrupt return. System stack pointer data is maintained even when the power of the unit is switched OFF. The user stack pointer (USP) is predecremented by the PUSH command and postincremented by the POP command regardless of system conditions.

- **Index Register (X, Y, Z) 16-bit**

The X-register and Z-register have virtually the same function, and can be used as 16-bit data pointers. Memory addresses to which a bias value of ±256 has been added can also be specified for these index registers.
Index registers are also used in conjunction with transfer commands and search commands. The Y register is used as the terminal for transfer commands and search commands only.

Flag Registers

MSB LSB

Z	C	LZ	UZ	SW	APO	*	*

- **Zero Flag Z (Non Zero Flag NZ)**

Reset to 0 when all the bits of a calculation result are 0 (Z), and set to 1 when data are present (NZ).

- **Carry Flag C (Non Carry Flag NC)**

Set to 1 when a carry or borrow occurs (C), and reset to 0 when a carry or borrow does not occur (NC).

- **Lower Digit Zero Flag LZ**

Reset to 0 when the low-order 4 bits are 0 due to a calculation result (LZ), and set to 1 when data are present.

- **Upper Digit Zero Flag UZ**

Reset to 0 when the high-order 4 bits are 0 (UZ), and set to 1 when data are present.

- **Power Switch State Flag SW**

Indicates the ON/OFF status of the power switch. This flag is set to 1 when power is ON, and reset to 0 when power is OFF.

- **Auto Power Off State Flag APO**

Set to 1 when an OFF command is executed while the power switch is ON, and reset to 0 when the power switch is OFF.

Status Registers

The status registers are used to determine the status of a variety of functions.

1. Interrupt Enable Register IE (Read/Write Enable) 8-bit

Performs interrupt masking and sets the interrupt conditions.

MSB □ INT 1 Interrupt enable (1: enable) ↑ High priority
□ KEY/Pulse interrupt enable
□ INT 2 interrupt enable
□ One-minute timer interrupt enable
□ ON interrupt enable
□ Power ON from ON terminal enable (1: enable)
□ INT 1 terminal interrupt edge specification 0: trailing edge, 1: leading edge
LSB □ INT 2 terminal interrupt level specification 0: low level, 1: high level

The RESET operation clears this register entirely, and bits 0, 1, 5, 6, and 7 are also cleared when power is OFF. The settings of bits 2 through 4 are maintained when power is switched OFF.

2. Interrupt Select and Key Output Regisfer IA (Read/Write Enable) 8-bit

Sets the type of interrupt and key output.

MSB □ Key interrupt (1) or pulse interrupt (0) specification
□ 32Hz (1) or 256Hz (0) specification when pulse interrupt is specified
□ } 2-bit pin specification is as follows when key interrupt is specified:
□ }
0: No pin specification
1: One pin
2: Two pins
3: All pins

□ } 4-bit key output specification as follows:
□ }
□ }
LSB □ }
0 : Stop
1 ~ 12 : One-key output
13 : All-key output
14 : Undefined
15 : Undefined

3. High-Order Address Specification Register UA (Read/Write Enable) 8-bit

The 2 bits of this register are added to the PC, X, Y, Z, SSP and USP 16-bit registers to allow specification of an 18-bit address (banking).

MSB □ }
□ } Z-register high-order address specification (0 ~ 3)
□ }
□ } X-register/Y-register high-order address specification
□ }
□ } SSP/USP register high-order address specification
□ }
LSB □ } PC high-order address specification

The RESET operation clears this register entirely, and Z, X, Y and PC are also cleared when power is OFF. SSP and USP are maintained when power is switched OFF.

4. Display Driver Control Register (Write Only) 8-bit

Outputs a control signal when display data or commands are sent to the display driver.

MSB	□	VDD2
	□	CLOCK ON(1), OFF(0) of $\phi1$ and $\phi2$
	□	Not used
	□	CE4
	□	CE3
	□	CE2
	□	CE1
LSB	□	OP

Bit 5 is undefined, and set values (except that set in bit 6) are output from the pin according to negative logic.

5. Port Status Specification Register PE (Read/Write Enable) 8-bit

Specifies a port for either input or output.

MSB	□	Specified port 7 as output (1) or input (0)
	□	Specifies port 6 as output (1) or input (0)
	□	Specifies port 5 as output (1) or input (0)
	□	Specifies port 4 as output (1) or input (0)
	□	Specifies port 3 as output (1) or input (0)
	□	Specifies port 2 as output (1) or input (0)
	□	Specifies port 1 as output (1) or input (0)
LSB	□	Specifies port 0 as output (1) or input (0)

All bits are cleared (reset to input status) when the RESET operation is performed or when power is switched OFF.

6. Port Data Register PD (Read/Write Enable) 8-bit

The contents written in this register are output from the pins of ports specified for output.

MSB	□	Port 7 output data value
	□	Port 6 output data value
	□	Port 5 output data value
	□	Port 4 output data value
	□	Port 3 output data value
	□	Port 2 output data value
	□	Port 1 output data value
LSB	□	Port 0 output data value

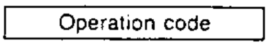
This register is not initialized by the RESET operation or when power is switched OFF (undefined).

2-3 COMMANDS

Command Length

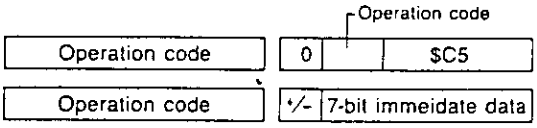
There are nine types of commands, classified according to length (number of bytes required).

- **1-Byte Commands**



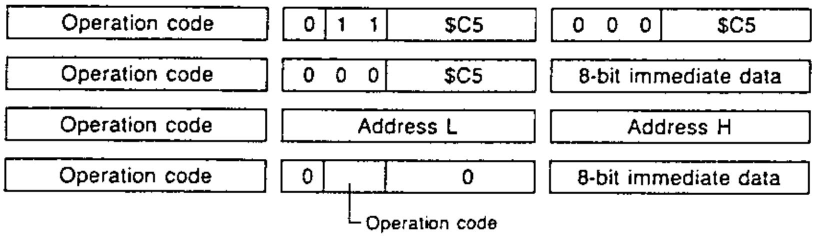
Operation code

- **2-Byte Commands**



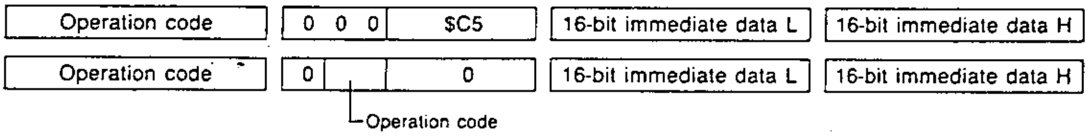
Operation code	0 $C5
Operation code	+/- 7-bit immeidate data

- **3-Byte Commands**



Operation code	0 1 1 $C5	0 0 0 $C5
Operation code	0 0 0 $C5	8-bit immediate data
Operation code	Address L	Address H
Operation code	0 0	8-bit immediate data

- **4-Byte Commands**



Operation code	0 0 0 $C5	16-bit immediate data L	16-bit immediate data H
Operation code	0 0	16-bit immediate data L	16-bit immediate data H

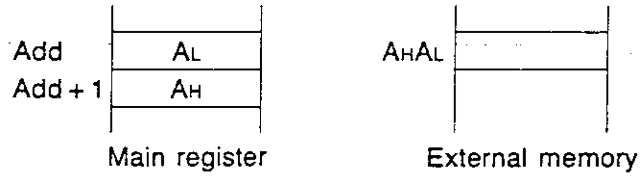
* $: main register address

Machine Language Command Symbols

The symbols listed below are used in the explanations and examples throughout this assembler reference:

$0 ~ 31 ($&H0 ~ $&H1F)	:	Main register address specification	
IY	:	Y-register (16-bit)	IR indicates both the X and Z registers. R-register: X-register or Z-register
IX	:	X-register (16-bit)	
IZ	:	Z-register (16-bit)	
SS	:	System stack pointer (16-bit)	
US	:	User stack pointer (16-bit)	
KY	:	Key input register (12-bit)	
PC	:	Program counter (16-bit)	
Z	:	Zero flag	
NZ	:	Non zero flag	
C	:	Carry flag	
NC	:	Non carry flag	
LZ	:	Low-order digit zero flag	
UZ	:	High-order digit zero flag	
IE	:	Interrupt enable register (8-bit)	
IA	:	Interrupt select register (8-bit)	Status registers
UA	:	High-order address specification register (8-bit)	
PD	:	Port data register (8-bit)	
PE	:	Port status specification register (8-bit)	
TM	:	Timer register (8-bit)	
C5	:	5-bit immediate data &H00 ~ &H1F or 0 ~ 31	
C8	:	8-bit immediate data &H00 ~ &HFF or 0 ~ 255	
C16	:	16-bit immediate data &H0000 ~ &HFFFF or 0 ~ 65535	
←	:	Transfer direction	
+	:	Addition	
−	:	Subtraction	
∧	:	Logical product (AND)	
∨	:	Logical sum (OR)	
⊕	:	Exclusive OR (XOR)	
&H	:	Hexadecimal	
( )	:	Contents of memory pointed to by the register included within ( )	

(IR ± $C5)	External memory contents addressed using R-register contents offset by main register contents
(IR ± C8)	External memory contents addressed using R-register contents offset by 8-bit immediate data
($C5)	External memory contents addressed using contents (16-bit) at location pointed to by main register Add (low-order), Add + 1 (high-order)



* (IR ± A): Either main register or 8-bit immediate data can be used to specify offset.

Flags

(Blank)	:	No change
0	:	Clear to 0
1	:	Set to 1
M	:	Set or clear according to command execution result

Command Components

Mnemonic

```
    ↓

    AD    $15,    $29
                   ↑
          ↑         2nd operand
         1st operand
```

MNEMONICS

TRANSFER COMMANDS (8-BIT)

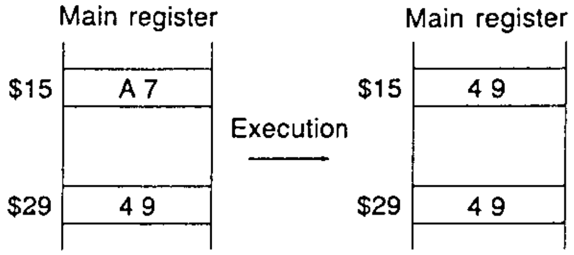
LD (LOAD)

• Main Register to Main Register

PURPOSE: Transfers the contents of the main register specified by the #2 operand to the main register specified by the #1 operand.

FORMAT: LD $C5, $C5

EXAMPLE: LD $15, $29

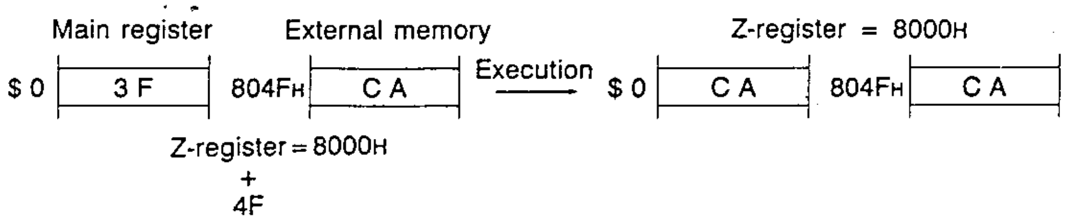


• External Memory to Main Register (1)

PURPOSE: Transfers the contents at the external memory location specified by the #2 operand to the main register specified by the #1 operand.

FORMAT: LD $C5, (IR ± A)

EXAMPLE: LD $0, (IZ + &H4F)
LD $15, (IX − $30)

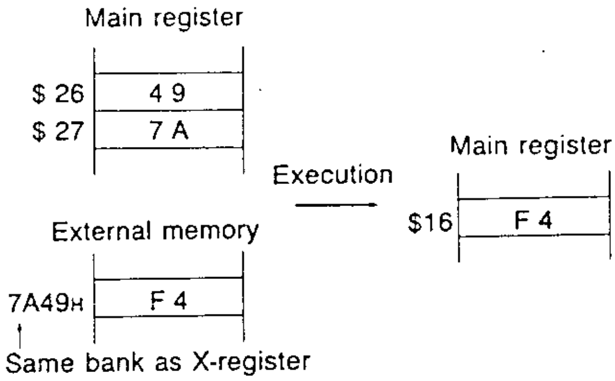


• External Memory to Main Register (2)

PURPOSE: Transfers to the main register specified by the #1 operand, the contents at the external memory location pointed to by the 2-byte main register specified by the #2 operand. The external memory bank (high-order address) is the same as the X-register bank.

FORMAT: LD $C5, ($C5)

EXAMPLE: LD $16, ($26)



- **Immediate Data to Main Register**

PURPOSE: Transfers the 8-bit immediate data contained in the #2 operand to the main register specified by the #1 operand.

FORMAT: LD $C5, C8

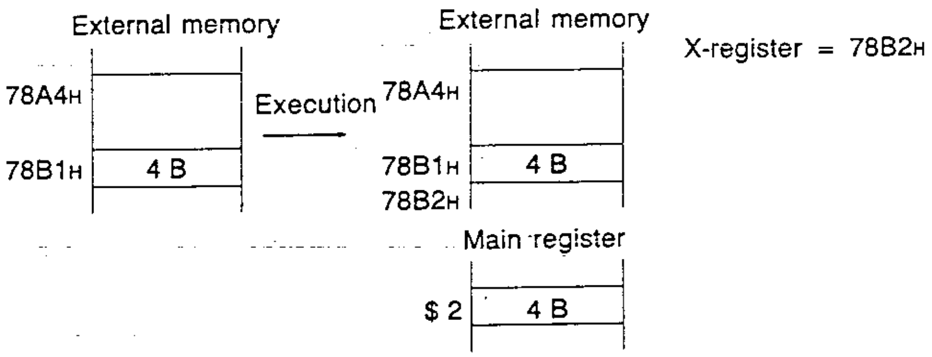
EXAMPLE: LD $10, &HA9

LDI (LOAD AND INCREMENT)

PURPOSE: Transfers the contents at the external memory location pointed to by the register specified by the #2 operand to the main register specified by the #1 operand. Then offset value + 1 is added to the register specified by the #2 operand.

FORMAT: LDI $12, (IR ± A)

EXAMPLE: LDI $15, (IX − $18)
LDI $02, (IX + &H0D)
When X-register = 78A4H



Execution example: LDI $02 (IX + &H0D)

ST (STORE)

• External Memory Specification Using R-Register

PURPOSE: Transfers the contents of the main register specified by the #1 operand to the external memory location specified by the #2 operand.

FORMAT: ST $C5, (IR ± A)

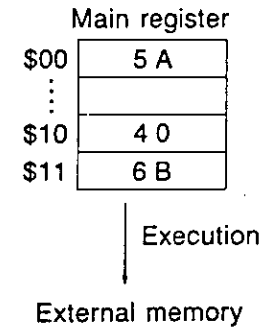
EXAMPLE:
```
ST  $11,  (IX - $04)
ST  $30,  (IZ + &HAC)
```

• External Memory Specification Using Main Register

PURPOSE: Transfers the contents of the main register specified by the #1 operand to the external memory location pointed to by the main register specified by the #2 operand ($C5 = low-order, $C5 + 1 = high-order). The external memory bank (high-order address) is the same as the X-register bank.

FORMAT: ST $C5, ($C5)

EXAMPLE: ST $00, ($10)



External memory address 6B40H = 5AH

STI (STORE AND INCREMENT)

PURPOSE: Transfers the contents of the main register specified by the #1 operand to the external memory location specified by the #2 operand. Then offset value + 1 is added to the R-register.

FORMAT: STI $C5, (IR ± A)

EXAMPLE:
```
STI  $31,  (IZ + 18)
STI  $0,  (IX - $05)
```
When X-register = 6A46H, register $05 = 03

Contents of main register $0 : 0 0 1 1 0 1 0 0

Execution

Contents of 6A46H − 3 (6A43H) : 0 0 1 1 0 1 0 0

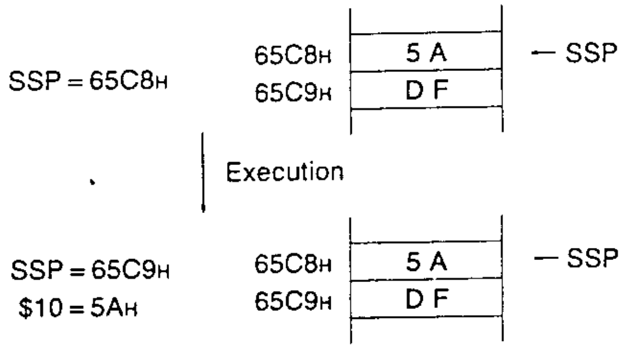
X-register changes to 6A44H

PPS (POP SYSTEM STACK POINTER)

PURPOSE: Transfers the contents of the external memory location pointed to by the SSP to the main register specified by the #1 operand. Then 1 is added to the SSP.

FORMAT: PPS $C5

EXAMPLE: PPS $10



PPU (POP USER STACK POINTER)

PURPOSE: Transfers the contents of the external memory location pointed to by the USP to the main register specified by the #1 operand. Then 1 is added to the USP (post increment).

FORMAT: PPU $C5

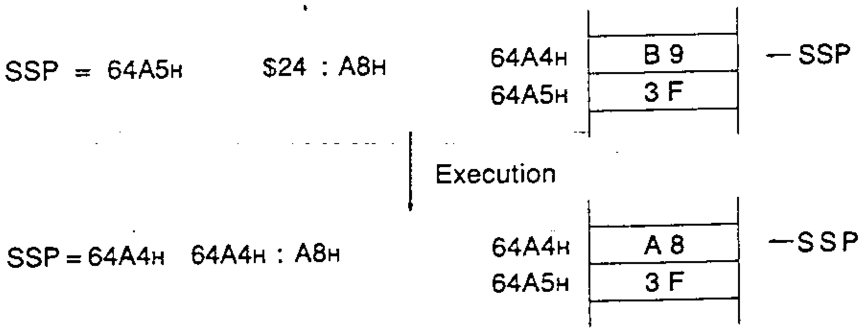
EXAMPLE: PPU $24

PHS (PUSH SYSTEM STACK POINTER)

PURPOSE: Subtracts 1 from the SSP (predecrement), then transfers the contents of the main register specified by the #1 operand to the external memory location pointed to by the SSP.

FORMAT: PHS $C5

EXAMPLE: PHS $24



PHU (PUSH USER STACK POINTER)

PURPOSE: Subtracts 1 from the USP (predecrement), then transfers the contents of the main register specified by the #1 operand to the memory location pointed to by the USP.

FORMAT: PHU $C5

EXAMPLE: PHU $12

GFL (GET FLAG)

PURPOSE: Transfers the flag register contents to the main register specified by the #1 operand.

FORMAT: GFL $C5

EXAMPLE: GFL $02

PFL (PUT FLAG)

PURPOSE: Transfers the contents of the main register specified by the #1 operand to the flag register (high-order 4 bits only).

FORMAT: PFL $C5

FLAGS: Z, C, LZ, and UZ flags. Each is changed to preset value.

EXAMPLE: PFL $15

GPO (GET PORT)

PURPOSE: Transfers port terminal contents to the main register specified by the #1 operand. The input value is transferred when input is specified, and the output value is transferred when output is specified.

FORMAT: GPO $C5

EXAMPLE: GPO $16

GST (GET STATUS)

PURPOSE: Transfers the contents of the status register specified by the #1 operand to the main register specified by the #2 operand.

FORMAT: GST Sreg. $C5 (Sreg = status register)

EXAMPLE: GST PE, $15
PD
UA
IE
TM

PE: $\boxed{1\ 0\ 0\ 1\ 1\ 0\ 1\ 1}$ $\xrightarrow{\text{Execution}}$ \$15 $\boxed{1\ 0\ 0\ 1\ 1\ 0\ 1\ 1}$

PE also retains its original contents.

PST (PUT STATUS)

- **Main Register**

PURPOSE: Transfers the contents of the main register specified by the #2 operand to the status register specified by the #1 operand.

FORMAT: PST Sreg. $C5 (Status registers except for Sreg. TM)

EXAMPLE: PST UA, $25

- **8-bit Immediate Data**

PURPOSE: Transfers the 8-bit immediate data included in the #2 operand to the status register specified by the #1 operand.

FORMAT: PST Sreg. C8 (Status registers except for Sreg. TM)

EXAMPLE: PST IE, &HF5

TRANSFER COMMANDS (16-BIT)

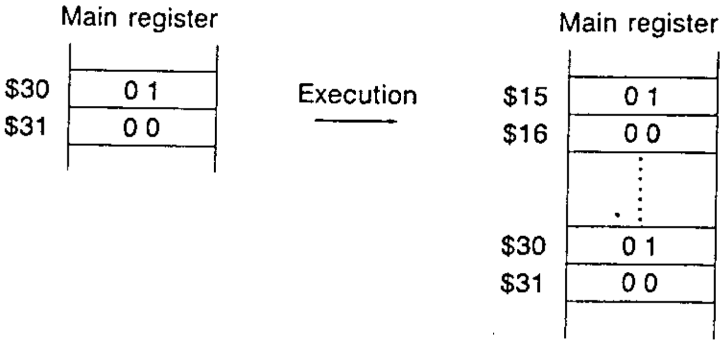
LDW (LOAD WORD)

• Main Register to Main Register

PURPOSE: Transfers the contents of the 2-byte main register specified by the #2 operand to the 2-byte main register specified by the #1 operand.

FORMAT: LDW $C5, $C5

EXAMPLE: LDW $15, $30

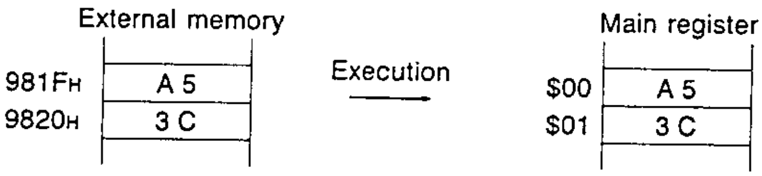


• External Memory to Main Register (1)

PURPOSE: Transfers the contents (2 bytes) at the memory location specified by the #2 operand to the 2-byte main register specified by the #1 operand.

FORMAT: LDW $C5, (IR ± $C5)

EXAMPLE: LDW $0, (IX + $15)



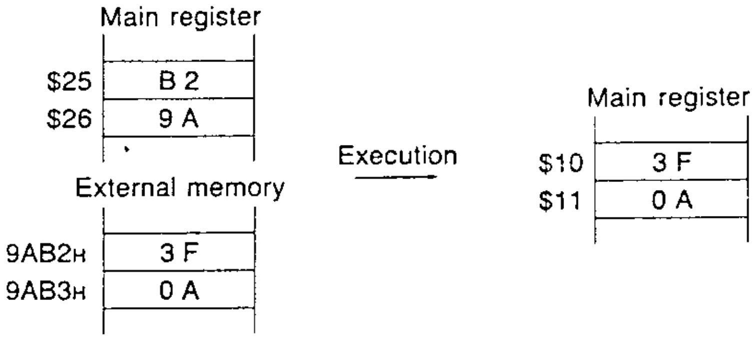
981CH+3
IX = 981CH
Register $15 = 03

- External Memory to Main Register (2)

PURPOSE: Transfers to the 2-byte main register specified by the #1 operand, the 2-byte contents at the external memory location pointed to by the 2-byte main register specified by the #2 operand. The external memory bank is the same as the high-order address of the X-register.

FORMAT: LDW $C5, ($C5)

EXAMPLE: LDW $10, ($25)

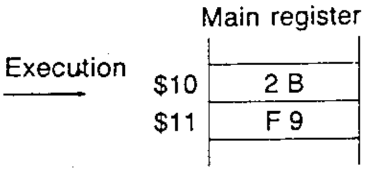


- 16-bit Immediate Data to Main Register

PURPOSE: Transfers the 16-bit immediate data contained in the #2 operand to the 2-byte main register specified by the #1 operand.

FORMAT: LDW $29, C16

EXAMPLE: LDW $10, &HF92B

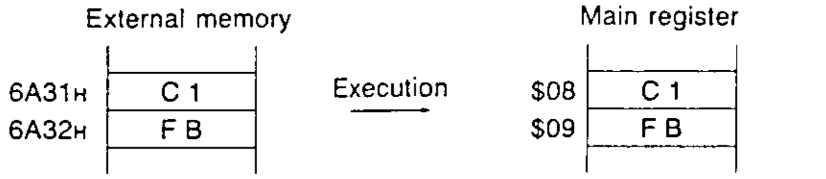


LDIW (LOAD WORD AND INCREMENT)

PURPOSE: Transfers to the 2-byte main register specified by the #1 operand, the contents at the external memory location pointed to by the register specified by the #2 operand. Then offset value + 2 is added to the register specified by the #2 operand (postincrement).

FORMAT: LDIW $C5, (IR ± $C5)

EXAMPLE: LDIW $08, (IZ − $30)



6A35H – 4
Z-register = 6A35H..........→ Z-register = 6A33H
Register $30 = 04 (Z-register ← Z-register – 4 + 2)

STW (STORE WORD)

• External Memory Specification Using R-register

PURPOSE: Transfers the contents of the 2-byte main register specified by the #1 operand to the 2-byte external memory location specified by the #2 operand.

FORMAT: STW $C5, (IR ± $C5)

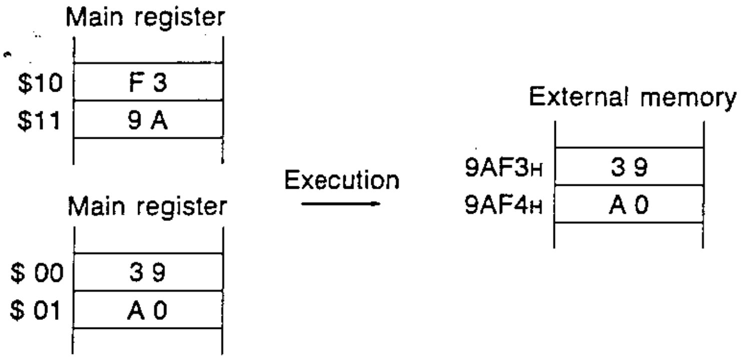
EXAMPLE: STW $11, (IX + $30)

• External Memory Specification Using Main Register

PURPOSE: Transfers the contents of the 2-byte register specified by the #1 operand to the 2-byte external memory location pointed to by the 2-byte main register specified by the #2 operand. The external memory bank conforms with the X-register.

FORMAT: STW $C5, ($C5)

EXAMPLE: STW $00, ($10)

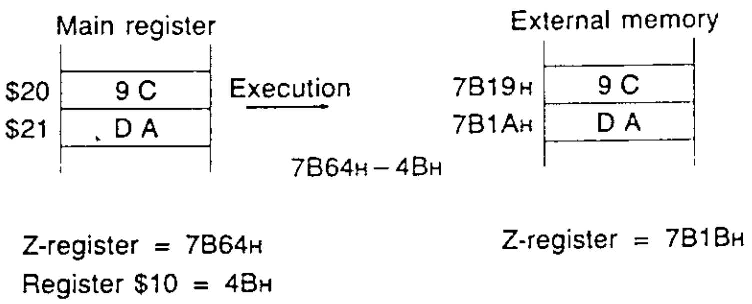


STIW (STORE WORD AND INCREMENT)

PURPOSE: Transfers the contents of the 2-byte main register specified by the # 1 operand to the 2-byte external memory location specified by the # 2 operand. Then the offset value + 2 is added to the R-register.

FORMAT: STIW $C5, (IR ± $C5)

EXAMPLE: STIW $20, (IZ − $10)

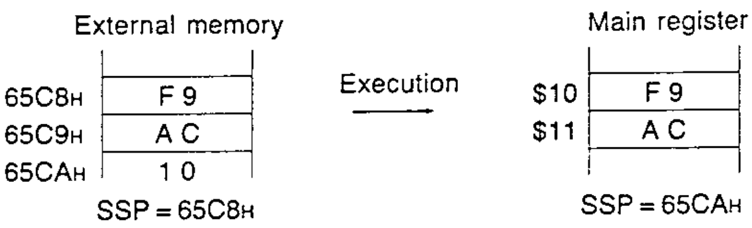


PPSW (POP SYSTEM STACK POINTER WORD)

PURPOSE: Transfers the contents of the external memory location pointed to by the SSP and SSP + 1 to the 2-byte main register specified by the # 1 operand. Then 2 is added to the SPP (postincrement).

FORMAT: PPSW $C5

EXAMPLE: PPSW $10



PPUW (POP USER STACK POINTER WORD)

PURPOSE: Transfers the contents of the external memory location pointed to by the USP and USP + 1 to the 2-byte main register specified by the # 1 operand.

FORMAT: PPUW $C5

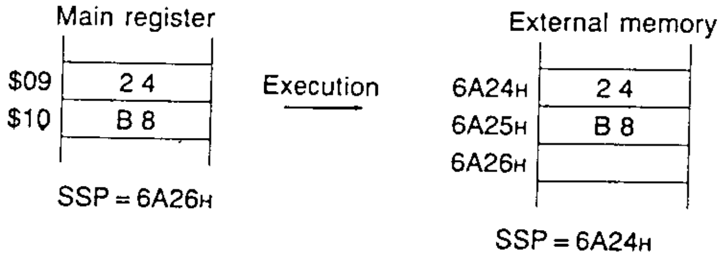
EXAMPLE: PPUW $15

PHSW (PUSH SYSTEM STACK POINTER WORD)

PURPOSE: Subtracts 1 (predecrement) from the SSP, then transfers the 2-byte contents of the main register address specified by the #1 operand ($C5, $C5 − 1) to the external memory location pointed to by the SSP (SSP, SSP − 1).

FORMAT: PHSW $C5

EXAMPLE: PHSW $10



PHUW (PUSH USER STACK POINTER WORD)

PURPOSE: Subtracts 1 (predecrement) from the USP, then transfers the 2-byte contents of the main register address specified by the #1 operand ($C5, $C5 − 1) to the external memory location pointed to by the USP (USP, USP − 1).

FORMAT: PHUW $C5

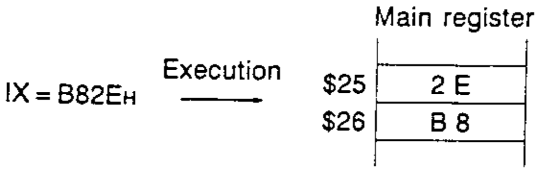
EXAMPLE: PHUW $15

GRE (GET. REGISTER)

PURPOSE: Transfers the contents of the 16-bit register specified by the #1 operand to the 2-byte main register specified by the #2 operand.

FORMAT: GRE Reg. $C5 (Reg. : IX, IY, IZ, SS, US, KY)

EXAMPLE: GRE IX, $25



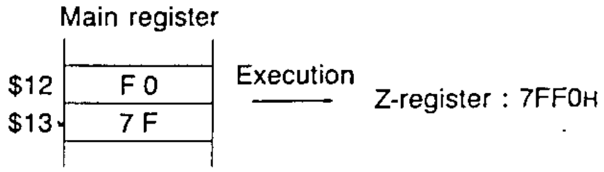
PRE (PUT REGISTER)

• Main Register to a 16-bit Register

PURPOSE: Transfers the contents of the 2-byte main register specified by the #2 operand to the 16-bit register specified by the #1 operand.

FORMAT: PRE Reg. $C5 (Reg: IX, IY, IZ, SS, US)

EXAMPLE: PRE IZ, $12



• 16-bit Immediate Data to a 16-bit Register

PURPOSE: Transfers the 16-bit immediate data contained in the #2 operand to the 16-bit register specified by the #1 operand.

FORMAT: PRE Reg. C16

EXAMPLE: PRE SS, &H70FF
SSP = 70FFH

ARITHMETIC COMMANDS (8-BIT)

AD (ADD)

• Main Register + Main Register → Main Register

PURPOSE: Adds the contents of the main register specified by the #2 operand to the contents of the main register specified by the #1 operand. The result is then written in the main register specified by the #1 operand.

FORMAT: AD $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: AD $10, $15

	Main register			Main register
$10	3 A	Execution	$10	9 B
$15	6 1		$15	6 1

Z = 1 (Result data present)
C = 0 (Non carry)
LZ = 1 (Low-order digit data present)
UZ = 1 (High-order digit data present)

9	B
1 0 0 1	1 0 1 1
High-order digit	Low-order digit

• Main Register + 8-bit Immediate Data → Main Register

PURPOSE: Adds the 8-bit immediate data contained in the #2 operand to the main register specified by the #1 operand. The result is then written in the main register specified by the #1 operand.

FORMAT: AD $C5, C8

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: AD $21, &HA4

- External Memory + Main Register → External Memory

PURPOSE: Adds the contents of the main register specified by the #2 operand to the contents of the external memory location pointed to by the main register specified by the #1 operand. The result is then written in the external memory location pointed to by the main register specified by the #1 operand.

FORMAT: AD (IR ± A), $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: AD (IX + $19), $0
AD (IZ – &H6B), $20

SB (SUBTRACT)

• Main Register – Main Register → Main Register

PURPOSE: Subtracts the contents of the main register specified by the #2 operand from the contents of the main register specified by the #1 operand. The result is then written in the main register specified by the #1 operand.

FORMAT: SB $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: SB $29, $03

	Main Register			Main register
$29	5 C	Execution →	$29	B 0
$03	A C		$03	A C

Z = 1, C = 1, LZ = 0, UZ = 1

• Main Register – 8-bit Immediate Data → Main Register

PURPOSE: Subtracts the 8-bit immediate data contained in the #2 operand from the contents of the main register specified by the #1 operand. The result is then written in the main register specified by the #1 operand.

FORMAT: SB $C5, C8

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: SB $27, 43 (Decimal)

• External Memory – Main Register → External Memory

PURPOSE: Subtracts the contents of the main register specified by the #2 operand from the contents of the external memory location pointed to by the the main register specified by the #1 operand. The result is then written in the external memory location pointed to by the main register specified by the #1 operand.

FORMAT: SB (IR ± A), $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: SB (IX – $05), $30
SB (IZ + $11), $29

ADB (BINARY CODED DECIMAL ADDITION)

• Main Register + Main Register → Main Register

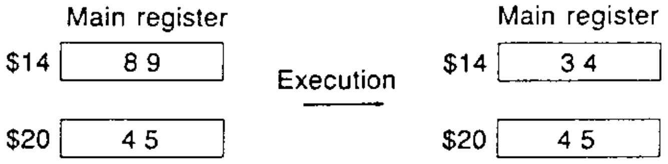
PURPOSE: Performs bcd addition of the contents of the main register specified by the #2 operand and the contents of the main register specified by the #1 operand. The result is then written in the main register specified by the #1 operand.

FORMAT: ADB $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: ADB $14, $20



Z = 1, C = 1, LZ = 1, UZ = 1

The high-order and low-order 4 bits are treated as binary coded decimal. In the above example, express as LD, $14, $H89 to set 89H to $14.

• Main Register + 8-bit Immediate Data → Main Register

PURPOSE: Performs bcd addition of the 8-bit immediate data contained in the #2 operand and the contents of the main register specified by the #1 operand. The result is then written in the main register specified by the #1 operand.

FORMAT: ADB $C5, C8

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: ADB $23, &H49

SBB (BINARY CODED DECIMAL SUBTRACTION)

• Main Register – Main Register → Main Register

PURPOSE: Performs bcd subtraction of the value of the main register specified by the #2 operand from the contents of the main register specified by the #1 operand. The result is then written in the main register specified by the #1 operand.

FORMAT: SBB $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: SBB $06, $23

	Main register			Main register
$06	3 5	Execution →	$06	9 9
$23	3 6		$23	3 6

Z = 1, C = 1, LZ = 1, UZ = 1

• Main Register – 8-bit Immediate Data → Main Register

PURPOSE: Performs bcd subtraction of the 8-bit immediate data contained in the #2 operand from the contents of the main register specified by the #1 operand. The result is then written in the main register specified by the #1 operand.

FORMAT: SBB $C5, C8

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: SBB $13, &H85

ADC (ADD CHECK)

• Main Register + Main Register

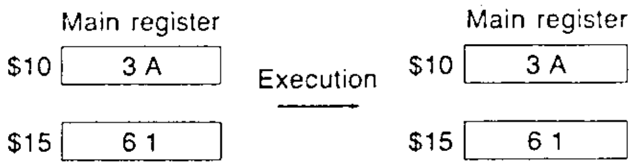
PURPOSE: Adds the contents of the main register specified by the #2 operand to the contents of the main register specified by the #1 operand. Only the status of the flags are changed and the result of the addition is not written anywhere.

FORMAT: ADC $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: ADC $10, $15



Contents of the main register do not change.

Z = 1, C = 0, LZ = 1, UZ = 1

- **Main Register + 8-bit Immediate Data**

PURPOSE: Adds the 8-bit immediate data contained in the #2 operand to the contents of the main register specified by the #1 operand. Only the status of the flags are changed and the result of the addition is not written anywhere.

FORMAT: ADC $C5, C8

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: ADC $12, 63

- **External Memory + Main Register**

PURPOSE: Adds the contents of the main register specified by the #2 operand to the contents of the external memory location pointed to by the #1 operand. Only the status of the flags are changed and the result of the addition is not written anywhere.

FORMAT: ADC (IR ± A), $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: ADC (IX + $21), $00
ADC (IZ – &H9E), $27

SBC (SUBTRACT CHECK)

• Main Register – Main Register

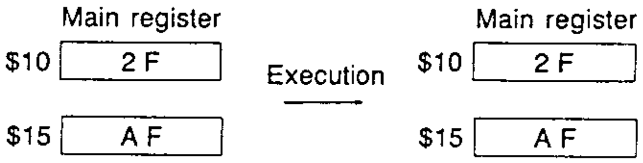
PURPOSE: Subtracts the contents of the main register specified by the #2 operand from the contents of the main register specified by the #1 operand. Only the status of the flags are changed and the result of the subtraction is not written anywhere.

FORMAT: SBC $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: SBC $10, $15



Data do not change.

(2FH – AFH) Z = 1, C = 1, LZ = 0, UZ = 1

• Main Register – 8-bit immediate data

PURPOSE: Subtracts the 8-bit immediate data contained in the #2 operand from the contents of the main register specified by the #1 operand. Only the status of the flags are changed and the result of the subtraction is not written anywhere.

FORMAT: SBC $C5, C8

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: SBC $24, 129

• External Memory – Main Register

PURPOSE: Subtracts the contents of the main register specified by the #2 operand from the contents of the external memory location pointed to by the #1 operand. Only the status of the flags are changed and the result of the subtraction is not written anywhere.

FORMAT: SBC (IR ± A), $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: SBC (IX – &H9B), $10
SBC (IZ + $11), $21

AN (AND)

• Main Register ∧ Main Register → Main Register

PURPOSE: Produces the logical product (AND) for the contents of the main register specified by the #1 operand and the contents of the main register specified by the #2 operand. The result is then written in the main register specified by the #1 operand.

FORMAT: AN $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	0	M	M

EXAMPLE: AN $12, $29

	Main register			Main register
$12	1 1 0 1 1 0 0 1	Execution →	$12	0 1 0 1 0 0 0 0
$29	0 1 1 1 0 1 1 0		$29	0 1 1 1 0 1 1 1

Z = 1, C = 0, LZ = 0, UZ = 1

• Main Register ∧ 8-bit Immediate Data → Main Register

PURPOSE: Produces the logical product (AND) for the contents of the main register specified by the #1 operand and the 8-bit immediate data contained in the #2 operand. The result is then written in the main register specified by the #1 operand.

FORMAT: AN $C5, C8

FLAGS:

Z	C	LZ	UZ
M	0	M	M

EXAMPLE: AN $20, &H3F

NA (NAND)

- **Main Register ∧ Main Register → Main Register**

PURPOSE: Takes the NAND (inverted AND) for the contents of the main register specified by the #1 operand and the contents of the main register specified by the #2 operand. The result is then written in the main register specified by the #1 operand.

FORMAT: NA $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	1	M	M

EXAMPLE: NA $11, $26

	Main register			Main register
$11	1 1 0 1 1 0 0 1	Execution	$11	1 0 1 0 1 1 1 0
$26	0 1 1 1 0 1 1 1	——→	$26	0 1 1 1 0 1 1 1

Z = 1, C = 1, LZ = 1, UZ = 1

- **Main register ∧ 8-bit Immediate Data → Main Register**

PURPOSE: Takes the NAND (inverted AND) for the contents of the main register specified by the #1 operand and the 8-bit immediate data contained in the #2 operand. The result is then written in the main register specified by the #1 operand.

FORMAT: NA $C5, C8

FLAGS:

Z	C	LZ	UZ
M	1	M	M

EXAMPLE: NA $00, 217

OR (OR)

- **Main Register ∨ Main Register → Main Register**

PURPOSE: Produces the logical sum (OR) for the contents of the main register specified by the #1 operand and the contents of the main register specified by the #2 operand. The result is then written in the main register specified by the #1 operand.

FORMAT: OR $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	1	M	M

EXAMPLE: OR $16, $10

	Main register			Main register
$16	0 0 1 0 1 0 0 1	Execution	$16	1 0 1 0 1 1 0 1
$10	1 0 0 0 1 1 0 0		$10	1 0 0 0 1 1 0 0

Z = 1, C = 1, LZ = 1, UZ = 1

- **Main Register ∨ 8-bit Immediate Data → Main Register**

PURPOSE: Produces the logical sum (OR) for the contents of the main register specified by the #1 operand and the 8-bit immediate data contained in the #2 operand. The result is then written in the main register specified by the #1 operand.

FORMAT: OR $C5, C8

FLAGS:

Z	C	LZ	UZ
M	1	M	M

EXAMPLE: OR $22, &HF1

XR (EXCLUSIVE OR)

- **Main Register ⊕ Main Register → Main Register**

PURPOSE: Takes the XOR (exclusive OR) for the contents of the main register specified by the #1 operand and the contents of the main register specified by the #2 operand. The result is then written in the main register specified by the #1 operand.

FORMAT: XR $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	0	M	M

EXAMPLE: XR $4, $16

	Main register			Main register
$4	1 0 1 1 0 1 0 0	Execution	$4	1 1 0 1 0 1 0 1
$16	0 1 1 0 0 0 0 1		$16	0 1 1 0 0 0 0 1

Z = 1; C = 0, LZ = 1, UZ = 1

- **Main register 8-bit ⊕ Immediate Data → Main Register**

PURPOSE: Takes the XOR (exclusive OR) for the contents of the main register specified by the #1 operand and the 8-bit immediate data contained in the #2 operand. The result is then written in the main register specified by the #1 operand.

FORMAT: XR $C5, C8

FLAGS:

Z	C	LZ	UZ
M	0	M	M

EXAMPLE: XR $15, 23

ANC (AND CHECK)

• Main Register ∧ Main Register

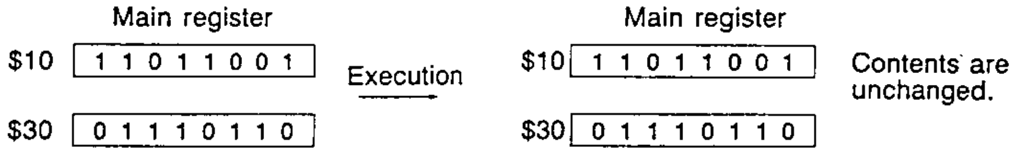
PURPOSE: Produces the logical product (AND) for the contents of the main register specified by the #1 operand and the contents of the main register specified by the #2 operand. Only the status of the flags are changed and the result of the AND is not written anywhere.

FORMAT: ANC $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	0	M	M

EXAMPLE: ANC $10, $30



Z=1, C=0, LZ=0, UZ=1

• Main Register ∧ 8-bit Immediate Data

PURPOSE: Produces the logical product (AND) for the contents of the main register specified by the #1 operand and the 8-bit immediate data contained in the #2 operand. Only the status of the flags are changed and the result of the AND is not written anywhere.

FORMAT: ANC $C5, C8

FLAGS:

Z	C	LZ	UZ
M	0	M	M

EXAMPLE: ANC $31, 2

NAC (NAND CHECK)

- **Main Register ∧ Main Register**

PURPOSE: Takes the NAND (inverted AND) for the contents of the main register specified by the #1 operand and the contents of the main register specified by the #2 operand. Only the status of the flags are changed and the result of the NAND is not written anywhere.

FORMAT: NAC $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	1	M	M

EXAMPLE: NAC $11, $26

	Main register			Main register	
$11	1 1 0 1 1 0 0 1	Execution →	$11	1 1 0 1 1 0 0 1	Contents are unchanged.
$26	0 1 1 1 0 1 1 1		$26	0 1 1 1 0 1 1 1	

Z = 1, C = 1, LZ = 1, UZ = 1

- **Main register ∧ 8-bit Immediate Data**

PURPOSE: Takes the NAND (inverted AND) for the contents of the main register specified by the #1 operand and the 8-bit immediate data contained in the #2 operand. Only the status of the flags are changed and the result is not written anywhere.

FORMAT: NAC $C5, C8

FLAGS:

Z	C	LZ	UZ
M	1	M	M

EXAMPLE: NAC $30, &HBC

ORC (OR CHECK)

- **Main Register ∨ Main Register**

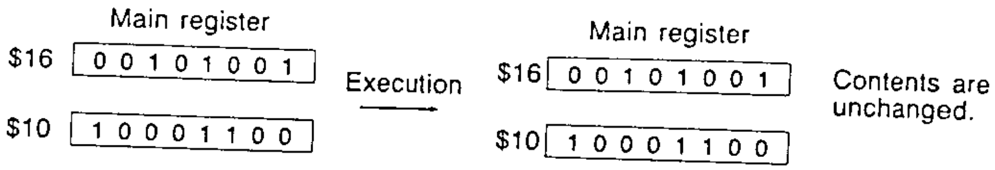
PURPOSE: Produces the logical sum (OR) for the contents of the main register specified by the #1 operand and the contents of the main register specified by the #2 operand. Only the status of the flags are changed and the result is not written anywhere.

FORMAT: ORC $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	1	M	M

EXAMPLE: ORC $16, $10



Z = 1, C = 1, LZ = 1, UZ = 1

- **Main Register ∨ 8-bit Immediate Data**

PURPOSE: Produces the logical sum (OR) for the contents of the main register specified by the #1 operand and the 8-bit immediate data contained in the #2 operand. Only the status of the flags are changed and the result is not written anywhere.

FORMAT: ORC $C5, C8

FLAGS:

Z	C	LZ	UZ
M	1	M	M

EXAMPLE: ORC $10, &H9C

XRC (EXCLUSIVE OR CHECK)

- **Main Register ⊕ Main Register**

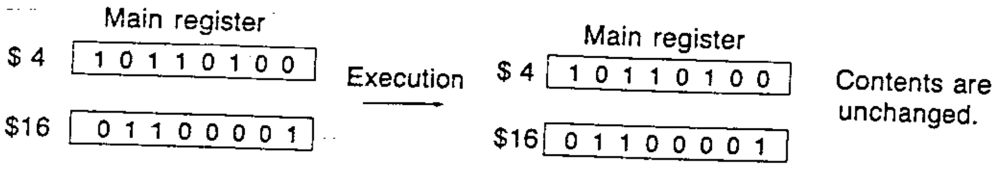
PURPOSE: Takes the XOR (exclusive OR) for the contents of the main register specified by the #1 operand and the contents of the main register specified by the #2 operand. Only the status of the flags are changed and the result is not written anywhere.

FORMAT: XRC $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	0	M	M

EXAMPLE: XRC $4, $16



Z = 1, C = 0, LZ = 1, UZ = 1

- **Main register ⊕ 8-bit Immediate Data**

PURPOSE: Takes the XOR (exclusive OR) for the contents of the main register specified by the #1 operand and the 8-bit immediate data contained in the #2 operand. Only the status of the flags are changed and the result is not written anywhere.

FORMAT: XRC $C5, C8

FLAGS:

Z	C	LZ	UZ
M	0	M	M

EXAMPLE: XRC $15, &HF0

ARITHMETIC COMMANDS (16-BIT)

ADW (ADD WORD)

- **Main Register + Main Register → Main Register**

PURPOSE: Adds the contents of the 2-byte main register specified by the #2 operand to the contents of the 2-byte main register specified by the #1 operand. The result is then written in the 2-byte main register specified by the #1 operand.

FORMAT: ADW $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

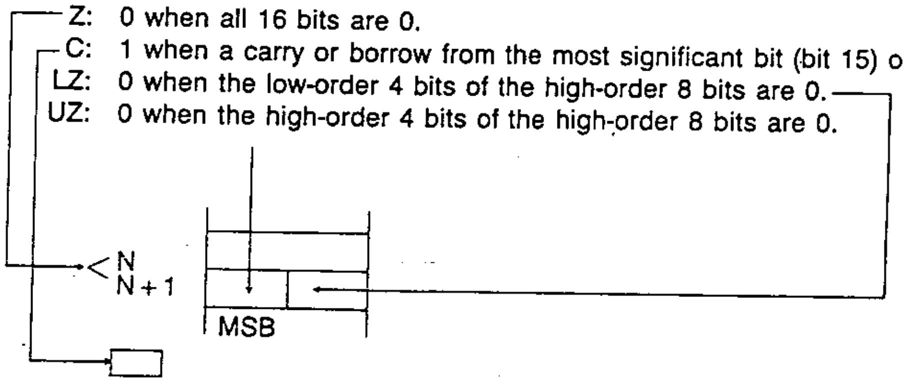
EXAMPLE: ADW $10, $12

	Main register			Main register
$10	6C		$10	5A
$11	A3	Execution →	$11	0E
$12	EE		$12	EE
$13	6A		$13	6A

Z = 1, C = 1, LZ = 1, UZ = 0

NOTE: 16-bit Arithmetic Flags

Z: 0 when all 16 bits are 0.
C: 1 when a carry or borrow from the most significant bit (bit 15) occurs.
LZ: 0 when the low-order 4 bits of the high-order 8 bits are 0.
UZ: 0 when the high-order 4 bits of the high-order 8 bits are 0.



Carry flag register

- **External Memory + Main Register → External Memory**

PURPOSE: Adds the contents of the 2-byte main register specified by the #2 operand to the contents of the 2-byte external memory location pointed to by the main register specified by the #1 operand. The result is then written in the 2-byte external memory location pointed to by the main register specified by the #1 operand.

FORMAT: ADW (IR ± $C5), $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: ADW (IX − $3), $18

	External memory			External memory
93F4H	2C		93F4H	58
93F5H	96	Execution →	93F5H	2C
	Main register			Main register
$18	2C		$18	2C
$19	96		$19	96

IX = 93F4H
$3 : 0

Z = 1, C = 1, LZ = 1, UZ = 1

SBW (SUBTRACT WORD)

- **Main Register − Main Register → Main Register**

PURPOSE: Subtracts the contents of the 2-byte main register specified by the #2 operand from the contents of the 2-byte main register specified by the #1 operand. The result is then written in the 2-byte main register specified by the #1 operand.

FORMAT: SBW $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: SBW $10, $28

	Main register			Main register
$10	4A		$10	21
$11	DB	Execution →	$11	00
$28	29		$28	29
$29	DB		$29	DB

Z = 1, C = 0, LZ = 0, UZ = 0

- External Memory − Main Register → External Memory

PURPOSE: Subtracts the contents of the 2-byte main register specified by the #2 operand from the contents of the 2-byte external memory location pointed to by the 2-byte main register specified by the #1 operand. The result is then written in the 2-byte external memory location pointed to by the main register specified by the #1 operand.

FORMAT: SBW (IR ± $5), $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: SBW (IZ + $19), $14

ADBW (BINARY CODED DECIMAL WORD ADDITION)

PURPOSE: Performs bcd addition of the contents of the 2-byte main register specified by the #2 operand to the contents of the 2-byte main register specified by the #1 operand. The result is then written in the 2-byte main register specified by the #1 operand.

FORMAT: ADBW $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: ADBW $16, $21

	Main register			Main register
$16	75		$16	39
$17	38	Execution →	$17	80
$21	64		$21	64
$22	41		$22	41

Z = 1, C = 0, LZ = 0, UZ = 1

SBBW (BINARY CODED DECIMAL WORD SUBTRACTION)

PURPOSE: Performs bcd subtraction of the contents of the 2-byte main register specified by the #2 operand from the value of the 2-byte main register specified by the #1 operand. The result is then written in the 2-byte main register specified by the #1 operand.

FORMAT: SBBW $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: SBBW $03, $29

ADCW (ADD CHECK WORD)

• Main Register + Main Register

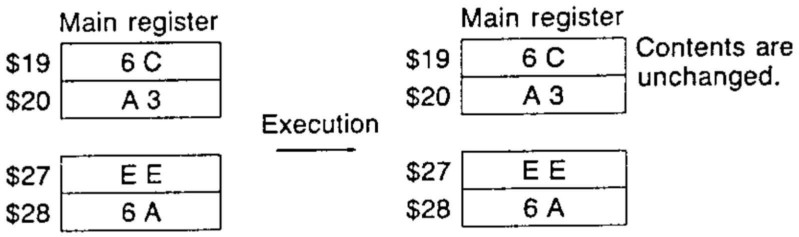
PURPOSE: Adds the contents of the 2-byte main register specified by the #2 operand to the contents of the 2-byte main register specified by the #1 operand. Only the status of the flags are changed and the result of the addition is not written anywhere.

FORMAT: ADCW $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: ADCW $19, $27



Z = 1, C = 1, LZ = 1, UZ = 0

• External Memory + Main Register

PURPOSE: Adds the contents of the 2-byte main register specified by the #2 operand to the contents of the 2-byte external memory location pointed to by the #1 operand. Only the status of the flags are changed and the result of the addition is not written anywhere.

FORMAT: ADCW (IR ± $C5), $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: ADCW (IX + $9), $14

ANW (AND WORD)

PURPOSE: Produces the logical product (AND) for the contents of the 2-byte main register specified by the #1 operand and the contents of the 2-byte main register specified by the #2 operand. The result is then written in the 2-byte main register specified by the #1 operand.

FORMAT: ANW $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	0	M	M

EXAMPLE: ANW $12, $19

	Main register MSB LSB			Main register MSB LSB
$12	1 0 0 1 1 1 0 0		$12	1 0 0 0 1 0 0 0
$13	1 1 0 0 0 1 0 1	Execution →	$13	0 1 0 0 0 1 0 0
$19	1 1 0 0 1 0 1 1		$19	1 1 0 0 1 0 1 1
$20	0 1 1 0 1 1 0 0		$20	0 1 1 0 1 1 0 0

Z = 1, C = 0, LZ = 1, UZ = 1

NAW (NAND WORD)

PURPOSE: Takes the NAND (inverted AND) for the contents of the 2-byte main register specified by the #1 operand and the contents of the 2-byte main register specified by the #2 operand. The result is then written in the 2-byte main register specified by the #1 operand.

FORMAT: NAW $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	1	M	M

EXAMPLE: NAW $23, $28

	Main register			Main register
$23	1 0 0 1 0 1 1 1		$23	0 1 1 1 1 0 0 1
$24	0 1 0 0 0 1 1 0	Execution →	$24	1 1 1 1 1 0 1 1
$28	1 1-0 0 0 1 1 0		$28	1 1 0 0 0 1 1 0
$29	0 0 0 1 1 1 0 1		$29	0 0 0 1 1 1 0 1

Z = 1, C = 1, LZ = 1, UZ = 1

SBCW (SUBTRACT CHECK WORD)

• Main Register – Main Register

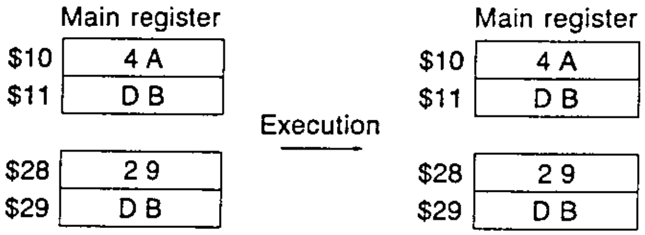
PURPOSE: Subtracts the contents of the 2-byte main register specified by the #2 operand from the contents of the 2-byte main register specified by the #1 operand. Only the status of the flags are changed and the result of the subtraction is not written anywhere.

FORMAT: SBCW $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: SBCW $10, $28



Z=1, C=0, LZ=0, UZ=0

• External Memory – Main Register

PURPOSE: Subtracts the contents of the 2-byte main register specified by the #2 operand from the contents of the 2-byte external memory location pointed to by the #1 operand. Only the status of the flags are changed and the result of the subtraction is not written anywhere.

FORMAT: SBCW (IR ± $C5), $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: SBCW (IZ – $15), $0

ORW (OR WORD)

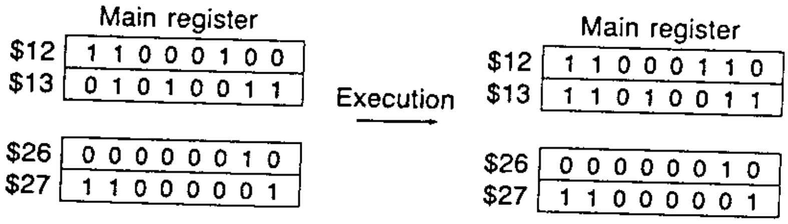
PURPOSE: Produces the logical sum (OR) for the contents of the 2-byte main register specified by the #1 operand and the contents of the 2-byte main register specified by the #2 operand. The result is then written in the 2-byte main register specified by the #1 operand.

FORMAT: ORW $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	1	M	M

EXAMPLE: ORW $12, $26



Z=1, C=1, LZ=1, UZ=1

XRW (EXCLUSIVE OR WORD)

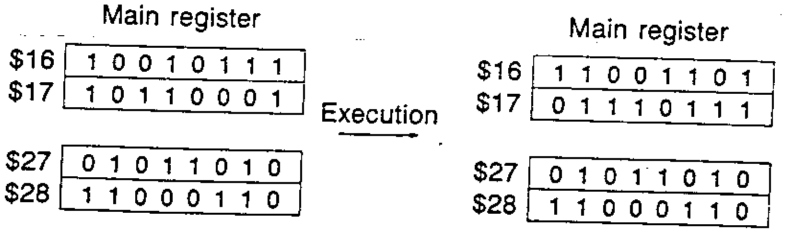
PURPOSE: Takes the XOR (exclusive OR) for the contents of the 2-byte main register specified by the #1 operand and the contents of the 2-byte main register specified by the #2 operand. The result is then written in the 2-byte main register specified by the #1 operand.

FORMAT: XRW $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	0	M	M

EXAMPLE: XRW $16, $27



Z=1, C=0, LZ=1, UZ=1

ANCW (AND CHECK WORD)

PURPOSE: Produces the logical product (AND) for the contents of the 2-byte main register specified by the #1 operand and the contents of the 2-byte main register specified by the #2 operand. Only the status of the flags are changed and the result of the AND is not written anywhere.

FORMAT: ANCW $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	0	M	M

EXAMPLE: ANCW $10, $21

NACW (NAND CHECK WORD)

PURPOSE: Takes the NAND (inverted AND) for the contents of the 2-byte main register specified by the #1 operand and the contents of the 2-byte main register specified by the #2 operand. Only the status of the flags are changed and the result of the NAND is not written anywhere.

FORMAT: NACW $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	1	M	M

EXAMPLE: NACW $00, $19

ORCW (OR CHECK WORD)

PURPOSE: Produces the logical sum (OR) for the contents of the 2-byte main register specified by the #1 operand and the contents of the 2-byte main register specified by the #2 operand. Only the status of the flags are changed and the result of the OR is not written anywhere.

FORMAT: ORCW $C5, $C5

FLAGS:

Z	C	LZ	UZ
M	1	M	M

EXAMPLE: ORCW $23, $02

XRCW (EXCLUSIVE OR CHECK WORD)

PURPOSE: Takes the XOR (exclusive OR) for the contents of the 2-byte main register specified by the #1 operand and the contents of the 2-byte main register specified by the #2 operand. Only the status of the flags are changed and the result is not written anywhere.

FORMAT: XRCW $C5, $C5

FLAGS:

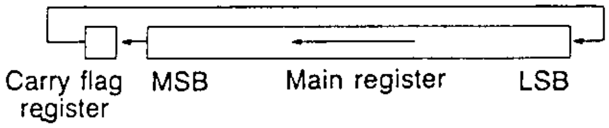
Z	C	LZ	UZ
M	0	M	M

EXAMPLE: XRCW $12, $30

ROTATE AND SHIFT COMMANDS(8-BIT)

ROU (ROTATE UP)

PURPOSE: Performs a left rotation between the main register specified by the #1 operand and the carry flag register.

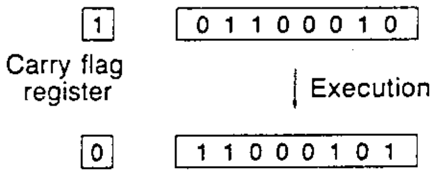


FORMAT: ROU $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

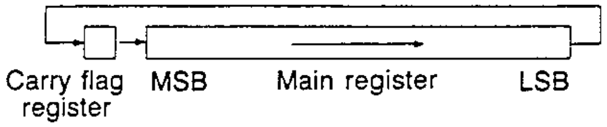
EXAMPLE: ROU $16



1 | 0 1 1 0 0 0 1 0
Carry flag register | Execution
0 | 1 1 0 0 0 1 0 1

ROD (ROTATE DOWN)

PURPOSE: Performs a right rotation between the main register specified by the #1 operand and the carry flag register.

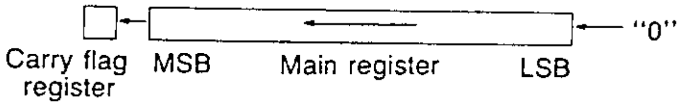


FORMAT: ROD $C5

EXAMPLE: ROD $18

BIU (BIT UP)

PURPOSE: Shifts the contents of the main register specified by the #1 operand to the left. The least significant bit receives a 0, while the data from the most significant bit moves to the carry flag register.

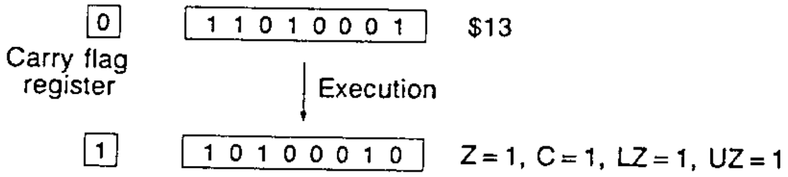


FORMAT: BIU $C5

FLAGS:

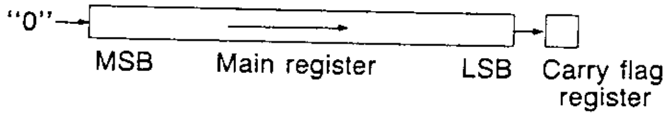
Z	C	LZ	UZ
M	M	M	M

EXAMPLE: BIU $13



BID (BIT DOWN)

PURPOSE: Shifts the contents of the main register specified by the #1 operand to the right. The most significant bit receives a 0, while the data from the least significant bit moves to the carry flag register.



FORMAT: BID $C5

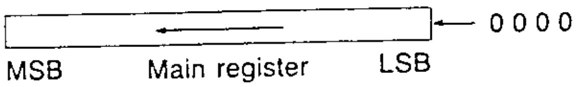
FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: BID $15

DIU (DIGIT UP)

PURPOSE: Shifts the contents of the main register specified by the #1 operand to the left in units of digits (4 bits). The low-order digit bits receive 0's.



FORMAT: DIU $C5

FLAGS:

Z	C	LZ	UZ
M	0	0	M

EXAMPLE: DIU $12

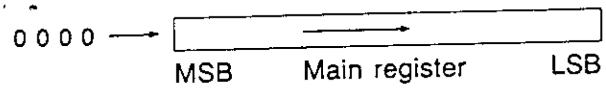
$12 [1 0 1 1 0 1 1 0]

↓ Execution

$12 [0 1 1 0 0 0 0 0]   Z = 1, C = 0, LZ = 0, UZ = 1

DID (DIGIT DOWN)

PURPOSE: Shifts the contents of the main register specified by the #1 operand to the right in units of digits (4 bits). The high-order digit bits receive 0's.



FORMAT: DID $C5

FLAGS:

Z	C	LZ	UZ
M	0	M	0

EXAMPLE: DID $19

INV (INVERT)

PURPOSE: Converts the contents of the main register specified by the #1 operand to their ones complement.

FORMAT: INV $C5

FLAGS:

Z	C	LZ	UZ
M	1	M	M

EXAMPLE: INV $11

```
$11 | 0 0 1 0 1 1 0 1 |
             |
             ↓ Execution
$11 | 1 1 0 1 0 0 1 0 |   Z=1, C=1, LZ=1, UZ=1
```

CMP (COMPLEMENT)

PURPOSE: Converts the contents of the main register specified by the #1 operand to their twos complement.

FORMAT: CMP $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

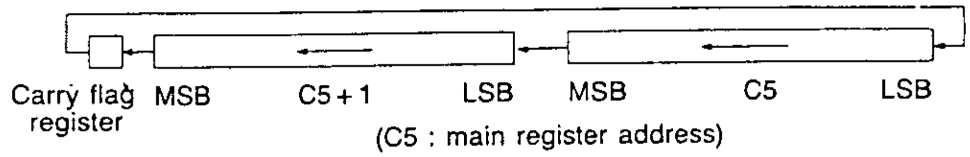
EXAMPLE: CMP $21

```
$21 | 0 1 0 1 1 0 1 0 |
             |
             ↓ Execution
$21 | 1 0 1 0 0 1 1 0 |   Z=1, C=0, LZ=1, UZ=1
```

ROTATE AND SHIFT COMMANDS(16-BIT)

ROUW (ROTATE UP WORD)

PURPOSE: Performs a left rotation between the 2-byte main register specified by the #1 operand and the carry flag register.



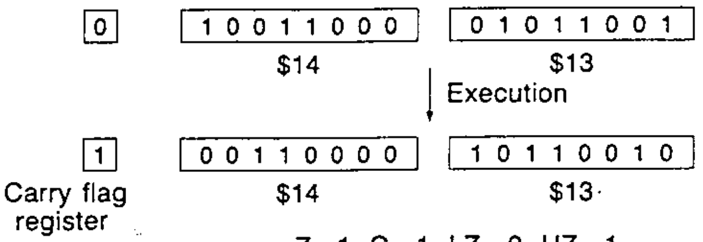
(C5 : main register address)

FORMAT: ROUW $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

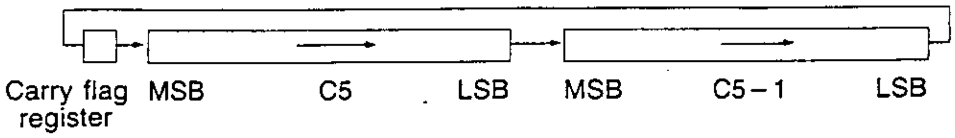
EXAMPLE: ROUW $13



Z=1, C=1, LZ=0, UZ=1

RODW (ROTATE DOWN WORD)

PURPOSE: Performs a right rotation between the 2-byte main register specified by the #1 operand and the carry flag register.

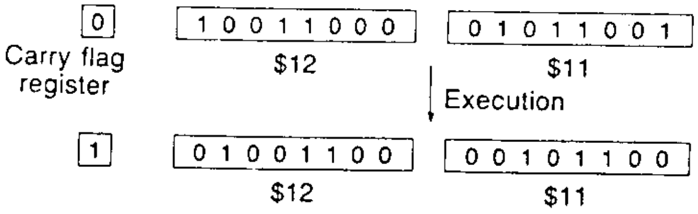


FORMAT: RODW $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

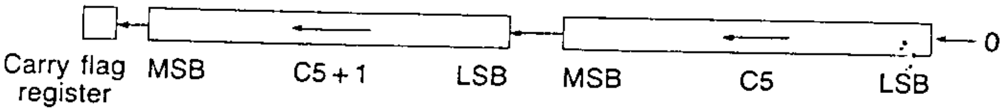
EXAMPLE: RODW $12



Z = 1, C = 1, LZ = 1, UZ = 1

BIUW (BIT UP WORD)

PURPOSE: Shifts the contents of the 2-byte main register specified by the #1 operand to the left. The least significant bit of the low-order byte receives a 0, while the data from the most significant bit of the high-order byte moves to the carry flag register.

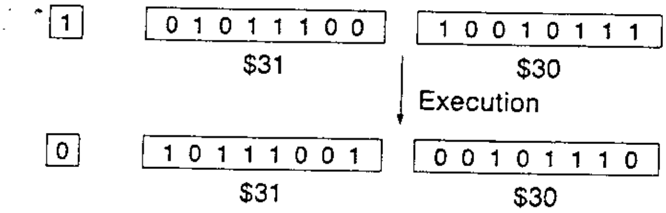


FORMAT: BIUW $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

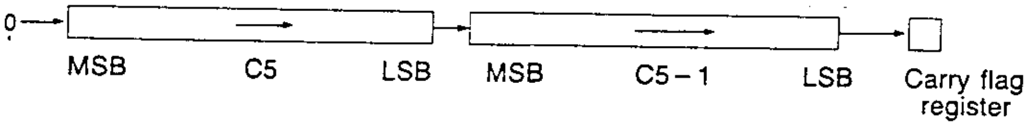
EXAMPLE: BIUW $30



Z = 1, C = 0, LZ = 1, UZ = 1

BIDW (BIT DOWN WORD)

PURPOSE: Shifts the contents of the 2-byte main register specified by the #1 operand to the right. The most significant bit of the high-order byte receives a 0, while the data from the least significant bit of the low-order byte moves to the carry flag register.

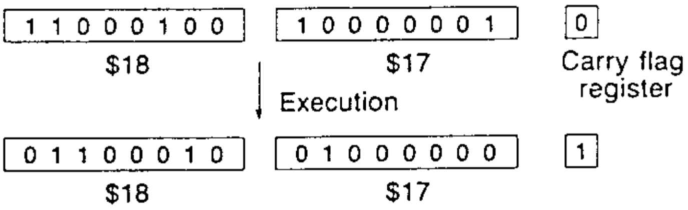


FORMAT: BIDW $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

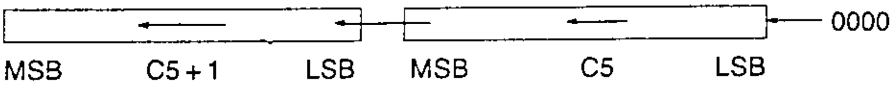
EXAMPLE: BIDW $18



Z = 1, C = 1, LZ = 0, UZ = 1

DIUW (DIGIT UP WORD)

PURPOSE: Shifts the contents of the 2-byte main register specified by the #1 operand to the left in units of digits (4 bits). The low-order digit bits of the low-order byte receive 0's.

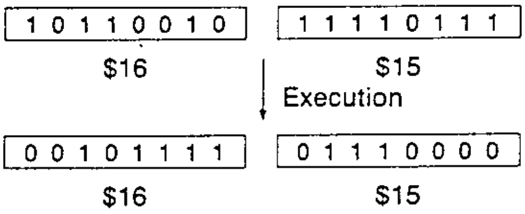


FORMAT: DIUW $C5

FLAGS:

Z	C	LZ	UZ
M	0	M	M

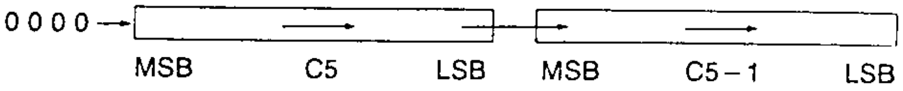
EXAMPLE: DIUW $15



Z = 1, C = 0, LZ = 1, UZ = 1

DIDW (DIGIT DOWN WORD)

PURPOSE: Shifts the contents of the 2-byte main register specified by the #1 operand to the right in units of digits (4 bits). The high-order digit bits of the high-order byte receive 0's.



FORMAT: DIDW $C5

FLAGS:

Z	C	LZ	UZ
M	0	M	M

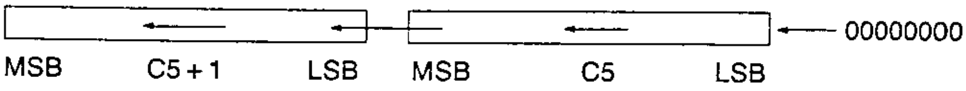
EXAMPLE: DIDW $09

```
[0 1 1 0 1 0 0 1]   [0 1 0 0 1 1 0 0]
       $09                 $08
                 | Execution
                 ↓
[0 0 0 0 0 1 1 0]   [1 0 0 1 0 1 0 0]
       $09                 $08
```

Z = 1, C = 0, LZ = 1, UZ = 1

BYUW (BYTE UP WORD)

PURPOSE: Shifts the contents of the 2-byte main register specified by the #1 operand to the left in units of bytes. The low-order byte receives 0's.



FORMAT: BYUW $C5

FLAGS:

Z	C	LZ	UZ
M	0	M	M

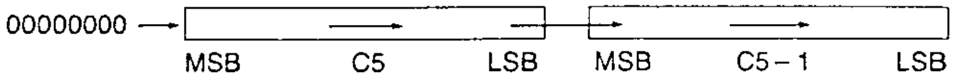
EXAMPLE: BYUW $15

```
[1 0 1 1 0 0 1 0]   [0 0 1 0 0 1 1 0]
       $16                 $15
                 | Execution
                 ↓
[0 0 1 0 0 1 1 0]   [0 0 0 0 0 0 0 0]
       $16                 $15
```

Z = 1, C = 0, LZ = 1, UZ = 1

BYDW (BYTE DOWN WORD)

PURPOSE: Shifts the contents of the 2-byte main register specified by the #1 operand to the right in units of bytes. The high-order byte receives 0's.



FORMAT: BYDW $C5

FLAGS:

Z	C	LZ	UZ
M	0	M	M

EXAMPLE: BYDW · $02

```
[1 1 0 1 0 0 1 0]  [0 1 0 0 1 0 1 0]
       $02                $01
                 | Execution
                 ↓
[0 0 0 0 0 0 0 0]  [1 1 0 1 0 0 1 0]
       $02                $01
```

Z = 1, C = 0, LZ = 1, UZ = 1

INVW (INVERT WORD)

PURPOSE: Converts the contents of the 2-byte main register specified by the #1 operand to their ones complement.

FORMAT: INVW· $C5

FLAGS:

Z	C	LZ	UZ
M	1	M	M

EXAMPLE: INVW $03

```
[1 0 1 1 0 0 0 1]  [0 1 0 1 0 0 1 1]
       $04                $03
                 | Execution
                 ↓
[0 1 0 0 1 1 1 0]  [1 0 1 0 1 1 0 0]
       $04                $03
```

Z = 1, C = 1, LZ = 1, UZ = 1

CMPW (COMPLEMENT WORD)

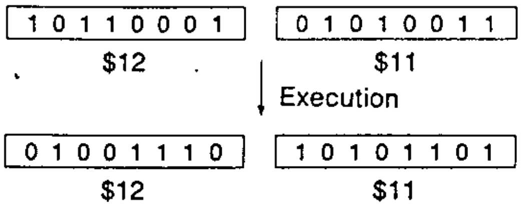
PURPOSE: Converts the contents of the 2-byte main register specified by the #1 operand to their twos complement.

FORMAT: CMPW $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: CMPW $11



Z = 1, C = 0, LZ = 1, UZ = 1

JUMP COMMANDS (ABSOLUTE)

Unconditional Jumps

JP (JUMP)

PURPOSE: Jumps to the address specified by the 16-bit immediate data contained in the #1 operand.

FORMAT: JP C16

EXAMPLE: JP &H984F

Conditional Jumps

JP Z (JUMP ON ZERO)

PURPOSE: Jumps to the address specified by the #2 operand when the zero flag (Z) is 0 (result = 0). Otherwise, execution proceeds to the next command.

FORMAT: JP Z, C16

EXAMPLE: JP Z, &HF3BC

JP NZ (JUMP ON NON-ZERO)

PURPOSE: Jumps to the address specified by the #2 operand when the zero flag (Z) is 1 (result≠0). Otherwise, execution proceeds to the next command.

FORMAT: JP NZ, C16

EXAMPLE: JP NZ, &H481F

JP C (JUMP ON CARRY)

PURPOSE: Jumps to the address specified by the #2 operand when the carry flag (C) is 1 (carry produced by result). Otherwise, execution proceeds to the next command.

FORMAT: JP C, C16

EXAMPLE: JP C, &HA34C

JP NC (JUMP ON NON-CARRY)

PURPOSE: Jumps to the address specified by the #2 operand when the carry flag (C) is 0 (no carry produced by result). Otherwise, execution proceeds to the next command.

FORMAT: JP NC, C16

EXAMPLE: JP NC, &H48FE

JP LZ (JUMP ON LOWER DIGIT ZERO)

PURPOSE: Jumps to the address specified by the #2 operand when the low-order digit zero flag is 0 (low- order 4 bits = 0). Otherwise, execution proceeds to the next command.

FORMAT: JP LZ, C16

EXAMPLE: JP LZ, &H79DA

JP UZ (JUMP ON UPPER DIGIT ZERO)

PURPOSE: Jumps to the address specified by the #2 operand when the high-order digit zero flag is 0 (high-order 4 bits = 0). Otherwise, execution proceeds to the next command.

FORMAT: JP UZ, C16

EXAMPLE: JP UZ, &HFF49

JUMP COMMANDS (RELATIVE)

With relative jump commands, a displacement expressed as immediate data (−127 ~ +127) is added to the program counter (PC), and a jump is executed to the resulting address. Adding a displacement to the program counter for branching is known as a "relative jump".

Unconditional Jumps

JR (RELATIVE JUMP)

PURPOSE: Performs a relative jump.

FORMAT: JR ± C7

7-bit immediate data

EXAMPLE: JR + &H7F

(127)

Conditional Jumps

JR Z (RELATIVE JUMP ON ZERO)

PURPOSE: Performs a relative jump when the zero flag is 0. Otherwise, execution proceeds to the next command.

FORMAT: JR Z, ±C7

EXAMPLE: JR Z, −18

JR NZ (RELATIVE JUMP ON NON-ZERO)

PURPOSE: Performs a relative jump when the zero flag (Z) is 1. Otherwise, execution proceeds to the next command.

FORMAT: JR NZ, ±C7

EXAMPLE: JR NZ, +&H4C

JR C (RELATIVE JUMP ON CARRY)

PURPOSE: Performs a relative jump when the carry flag (C) is 1. Otherwise, execution proceeds to the next command.

FORMAT: JR C, ±C7

EXAMPLE: JR C, −67

JR NC (RELATIVE JUMP ON NON-CARRY)

PURPOSE: Performs a relative jump when the non-carry flag (NC) is 0. Otherwise, execution proceeds to the next command.

FORMAT: JR NC, ±C7

EXAMPLE: JR NC, +120

JR LZ (RELATIVE JUMP ON LOWER DIGIT ZERO)

PURPOSE: Performs a relative jump when the low-order digit zero flag is 0 (low-order 4 bits = 0). Otherwise, execution proceeds to the next command.

FORMAT: JR LZ, ±C7

EXAMPLE: JR LZ, −&H7E

JR UZ (RELATIVE JUMP ON UPPER DIGIT ZERO)

PURPOSE: Performs a relative jump when the high-order digit zero flag is 0 (high-order 4 bits = 0). Otherwise, execution proceeds to the next command.

FORMAT: JR UZ, ±C7

EXAMPLE: JR UZ, +127

CALL COMMANDS

The program counter contents that indicate the final address of the CAL command currently being executed are pushed into the stack, and the system stack pointer is decremented (– 2). Then execution jumps to an address specified by 16-bit immediate data. The RTN command (see page 177) is used to return to the command following the original CAL command.

Unconditional Call

CAL (CALL)

PURPOSE: Calls the address specified by the 16-bit immediate data contained in the #1 operand.

FORMAT: CAL C16

EXAMPLE: CAL &HF42C

Conditional Calls

CAL Z (CALL ON ZERO)

PURPOSE: Calls the address specified by the 16-bit immediate data contained in the #2 operand when the zero flag (Z) is 0. Otherwise, execution proceeds to the next command.

FORMAT: CAL Z, C16

EXAMPLE: CAL Z, &H6E3F

CAL NZ (CALL ON NON-ZERO)

PURPOSE: Calls the address specified by the 16-bit immediate data contained in the #2 operand when the zero flag (Z) is 1. Otherwise, execution proceeds to the next command.

FORMAT: CAL NZ, C16

EXAMPLE: CAL NZ, &H963E

CAL C (CALL ON CARRY)

PURPOSE: Calls the address specified by the 16-bit immediate data contained in the #2 operand when the carry flag (C) is 1. Otherwise, execution proceeds to the next command.

FORMAT: CAL C, C16

EXAMPLE: CAL C, &H4920

CAL NC (CALL ON NON-CARRY)

PURPOSE: Calls the address specified by the 16-bit immediate data contained in the #2 operand when the carry flag (C) is 0. Otherwise, execution proceeds to the next command.

FORMAT: CAL NC, C16

EXAMPLE: CAL NC, &HABCD

CAL LZ (CALL ON LOWER DIGIT ZERO)

PURPOSE: Calls the address specified by the 16-bit immediate data contained in the #2 operand when the low-order digit zero flag is 0 (low-order 4 bits = 0). Otherwise, execution proceeds to the next command.

FORMAT: CAL LZ, C16

EXAMPLE: CAL LZ, &H4811

CAL UZ (CALL ON UPPER DIGIT ZERO)

PURPOSE: Calls the address specified by the 16-bit immediate data contained in the #2 operand when the high-order digit zero flag is 0 (high-order 4 bits = 0). Otherwise, execution proceeds to the next command.

FORMAT: CAL UZ, C16

EXAMPLE: CAL UZ, &H045F

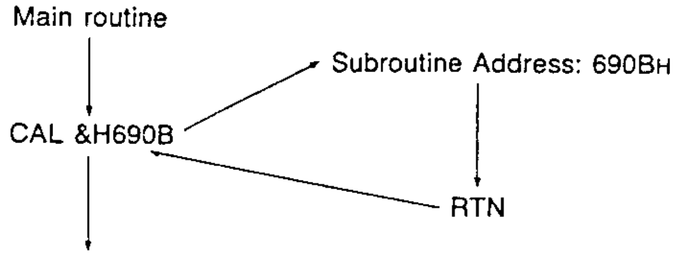
RETURN COMMANDS

The RETURN commands are used to return from a subroutine to the main routine. A 16-bit address is popped from the stack causing the system stack pointer to be incremented (+2). One is added to the address value and the result is transferred to the program counter (PC).

Unconditional Return

RTN (RETURN)

PURPOSE: Pops an address from the stack (SSP + 2) and transfers the result of address + 1 to the program counter.



FORMAT: RTN

EXAMPLE: RTN

Conditional Return

RTN Z (RETURN ON ZERO)

PURPOSE: Returns when the zero flag (Z) is 0. Otherwise, execution proceeds to the next command.

FORMAT: RTN Z

EXAMPLE: RTN Z

RTN NZ (RETURN ON NON-ZERO)

PURPOSE: Returns when the zero flag (Z) is 1. Otherwise, execution proceeds to the next command.

FORMAT: RTN NZ

EXAMPLE: RTN NZ

RTN C (RETURN ON CARRY)

PURPOSE: Returns when the carry flag (C) is 1. Otherwise, execution proceeds to the next command.

FORMAT: RTN C

EXAMPLE: RTN C

RTN NC (RETURN ON NON-CARRY)

PURPOSE: Returns when the carry flag (C) is 0. Otherwise, execution proceeds to the next command.

FORMAT: RTN NC

EXAMPLE: RTN NC

RTN LZ (RETURN ON LOWER DIGIT ZERO)

PURPOSE: Returns when the low-order digit zero flag is 0 (low-order 4 bits = 0). Otherwise, execution proceeds to the next command.

FORMAT: RTN LZ

EXAMPLE: RTN LZ

RTN UZ (RETURN ON UPPER DIGIT ZERO)

PURPOSE: Returns when the high-order digit zero flag is 0 (high-order 4 bits = 0). Otherwise, execution proceeds to the next command.

FORMAT: RTN UZ

EXAMPLE: RTN UZ

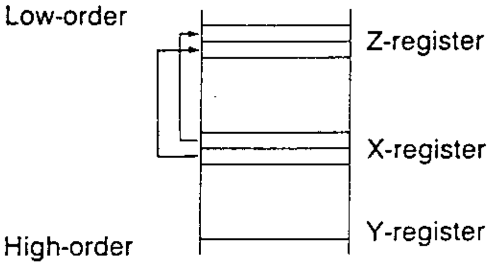
BLOCK MOVE COMMANDS

Blocks of memory can be moved from one location to another using the index register.

BUP (BLOCK UP)

PURPOSE: Moves a block of memory where:
X-register = original block beginning address
Y-register = original block ending address
Z-register = destination beginning address

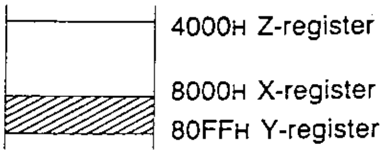
NOTE: In this case, X-register < Y-register.



FORMAT: BUP

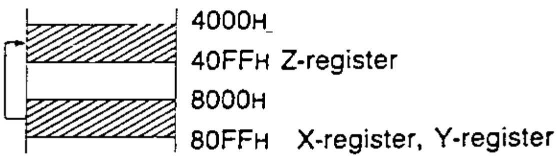
EXAMPLE: BUP

X-register = 8000H Y-register = 80FFH Z-register = 4000H



Execution

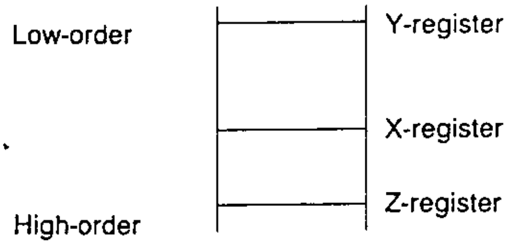
X-register = 80FFH Y-register = 80FFH Z-register = 40FFH



BDN (BLOCK DOWN)

PURPOSE: Moves a block of memory where:
X-register = original block beginning address
Y-register = original block ending address
Z-register = destination beginning address

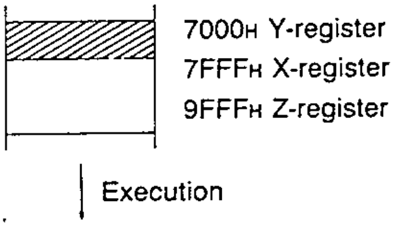
NOTE: In this case, X-register > Y-register.



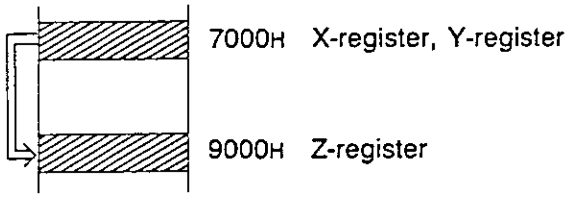
FORMAT: BDN

EXAMPLE: BDN

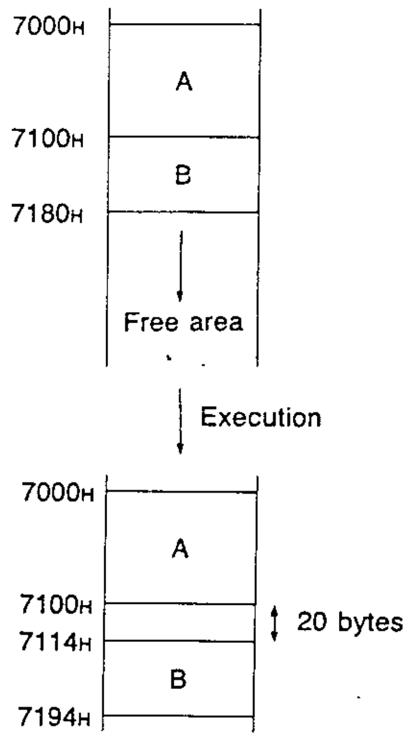
X-register : 7FFFH Y-register : 7000H Z-register : 9FFFH



X-register : 7000H Y-register : 7000H Z-register : 9000H



★ Block Move Command Applications



Blocks can be inserted within memory without deleting data that already are present. Assume that two blocks of data, A and B are present in memory as illustrated on the left. To insert a third block of data (C) which is 20 bytes (14H) in size, it is first required to open up a 20-byte area using the BDN block move command after making the following preparations:

X-register ← 717FH (PRE IX, &H717F)
Y-register ← 7100H (PRE IY, &H7100)
Z-register ← 717FH + 14H = 7193H
(PRE IZ, &H7193)

SEARCH COMMANDS

The search commands are used to locate specified internal memory contents or 8-bit immediate data within a range of memory defined by the index registers.

SUP (SEARCH UP)

• Internal Memory Value

PURPOSE: Searches the external memory within a specific range (X-register = beginning address and Y-register = ending address) for the contents of the main register specified by the #1 operand. The zero flag is reset (to 0) and the search is terminated when the data is located, while X-register = Y-register is set if the search is unsuccessful (Z flag = 1).
NOTE: In this case, X-register < Y-register.

FORMAT: SUP $C5

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: SUP $12

$12 $\boxed{4C}$ X-register = 7500H, Y-register = 7FFFH

↓ Execution

When 4C is located: Z = 0, C = 0, LZ = 0, UZ = 0
X-register = location (address) of 4C, Y-register = 7FFFH

• 8-bit Immediate Data

PURPOSE: Searches the external memory within a specific range (X-register = beginning address and Y-register = ending address) for the 8-bit immediate data contained in the #1 operand. The zero flag is reset (to 0) and the search is terminated when the data is located.
NOTE: In this case, X-register < Y-register.

FORMAT: SUP C8

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: SUP &H4C

SDN (SEARCH DOWN)

•Internal Memory Value

PURPOSE: Searches the external memory within a specific range (X-register = beginning address and Y-register = ending address − 1) for the contents of the main register specified by the #1 operand. The zero flag is reset (to 0) and the search is terminated when the data is located, while X-register = Y-register is set if the search is unsuccessful (Z flag = 1).
NOTE: In this case, X-register > Y-register.

FORMAT: SDN $C5

FLAGS:

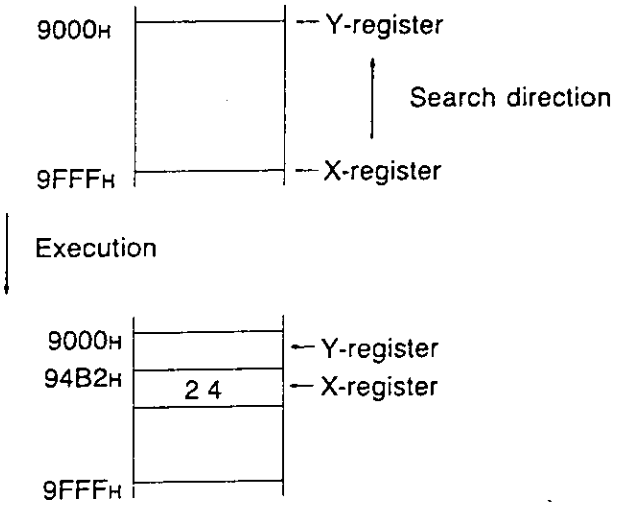
Z	C	LZ	UZ
M	M	M	M

EXAMPLE: SDN $15

Main register
$15 [2 4]

X-register = 9FFFH
Y-register = 9000H

When 24H is located at 94B2H:



Z = 0, C = 0, LZ = 0, UZ = 0
X-register = 94B2H, Y-register = 9000H

• 8-bit Immediate Data

PURPOSE: Searches the external memory within a specific range (X-register = beginning address and Y-register = ending address − 1) for the 8-bit immediate data contained in the #1 operand. The zero flag is reset (to 0) and the X-register is assigned a value which represents the address location − 1, while X-register = Y-register is set if the search is unsuccessful (Z flag = 1).
NOTE: In this case, X-register > Y-register.

FORMAT: SDN C8

FLAGS:

Z	C	LZ	UZ
M	M	M	M

EXAMPLE: SDN &H4A

SPECIAL COMMANDS

NOP (NO OPERATION)

PURPOSE: Increments (+1) the program counter.

FORMAT: NOP

EXAMPLE: NOP

CLT (CLEAR TIMER)

PURPOSE: Inputs a SET signal to all timer counters to set the value of the counters to 0.

FORMAT: CLT

EXAMPLE: CLT

FST (FAST)

PURPOSE: Uses the system clock without dividing (high-speed processing mode).

FORMAT: FST

EXAMPLE: FST

SLW (SLOW)

PURPOSE: Uses the system clock with 1/16 dividing (low power mode). However, automatic switching to the high-speed processing mode is performed in the interrupt handling routine.

FORMAT: SLW

EXAMPLE: SLW

OFF (OFF)

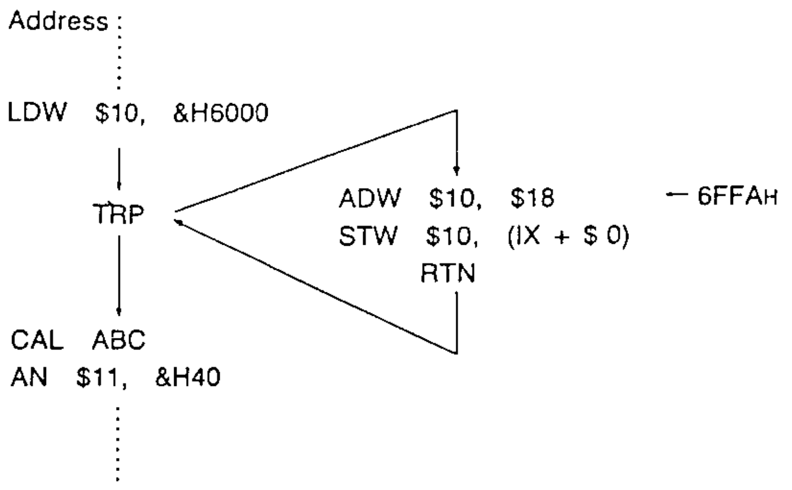
PURPOSE: Cuts the power supply of the internal logic VDD system.

FORMAT: OFF

EXAMPLE: OFF

TRP (TRAP)

PURPOSE: Input of the TRP command operation code (FF) causes the trap address (address where FF is written) to be entered into the stack at the present SSP location. Processing then performed from the fixed address 6FFAH, and is returned to the command following the TRP command by a RTN command.



```
Address :
LDW  $10,  &H6000
      TRP
                     ADW  $10,  $18          — 6FFAH
                     STW  $10,  (IX + $ 0)
                          RTN
CAL  ABC
AN  $11,  &H40
```

FORMAT: TRP

EXAMPLE: TRP

CANI (CANCEL INTERRUPT)

PURPOSE: Clears the hardware interrupt request latch which has the highest priority.

FORMAT: CANI

EXAMPLE: CANI

RTNI (RETURN FROM INTERRUPT)

PURPOSE: Loads the system stack contents into the program counter (PC), returns to the resulting address and adds 2 to the SSP. This command is used to return from an interrupt handling routine.

FORMAT: RTNI

EXAMPLE: RTNI

PART 3
SAMPLE PROGRAMS

Bit Shift Display

```
;       ASSEMBLER PROG.   "SHIFT"
        ORG    &H7000
        START  &H7000
;
DOTDI:  EQU    &H022C
;
        PRE    IX,&H6202
        PRE    IY,&H6800
        PRE    IZ,&H6201
        LD     $0,(IZ+0)
        BUP
        ST     $0,(IX+0)
        CAL    DOTDI           }
        RTN                    } JP DOTDI also O.K.

10 CLS
20 LOCATE 12,3:PRINT "AD-1990"
30 A$=INKEY$:IF A$="" THEN 30
40 CALL "SHIFT.EXE"
50 GOTO 30
```

EXECUTION: Bit shift left when key is pressed.

OPERATION: The above noted assembler source list is assembled, and a BASIC program is executed.

Subroutines for Reversed Display

```
:   ASSEMBLER PROG.  "REV"
        ORG    &H7000
        START  &H7000
DOTDI:  EQU    &H022C
        PRE    IX,&H6201
        LDW    $0,&H600
LOOP:   LD     $2,(IX+0)
        INV    $2
        STI    $2,(IX+0)
        SBW    $0,$30
        JR     NZ,LOOP
        CAL    DOTDI        } JP DOTDI also O.K.
        RTN                 }
```

```
10 CLS
20 LOCATE 12,3:PRINT "AD-1990"
30 A$=INKEY$:IF A$="" THEN 30
40 CALL "REV.EXE"
50 GOTO 30
```

EXECUTION: Screen reversed when key is pressed.

OPERATION: The above noted assembler source list is assembled, and a BASIC program is executed.

PART 4
MONITOR

MONITOR MODE

Outline

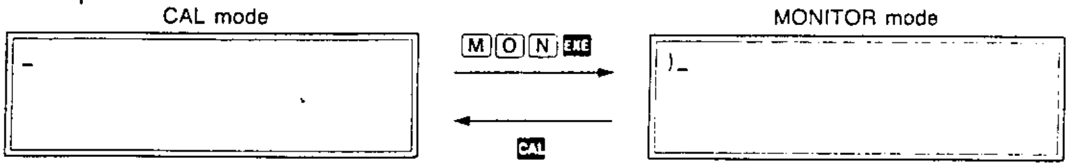
The MONITOR mode can be used to view or change memory contents.

Entering and Exiting the MONITOR Mode

a) The MONITOR mode can be entered by executing the MON command while in the CAL mode or BASIC mode.

b) Pressing the MENU key while in the MONITOR mode returns to the MENU mode, while pressing the CAL key returns to the CAL mode.

Example:



```
_
```

```
)_
```

Command Execution

B (BANK) : Switches the memory bank
D (DUMP): Memory content output
E (EDIT) : Memory content change

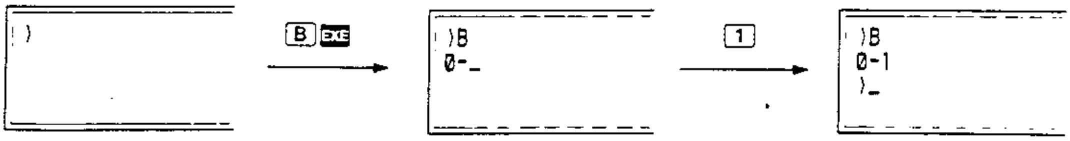
B: BANK SWITCH

PURPOSE: Sets (switches) the object bank for execution of the DUMP or EDIT command.

EXPLANATION:

1. Switches between BANK 0 and BANK 1.
2. The initialized setting is BANK 1.

EXAMPLE: Setting BANK 1 (from BANK 0):



```
)
```

```
)B
0-_
```

```
)B
0-1
)_
```

Following B EXE directly by EXE without any further input returns to command input stand by without switching the bank.

D: DUMP MEMORY

PURPOSE: Displays the memory contents.

FORMAT: D [display start address]

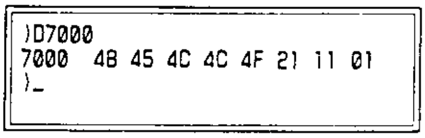
PARAMETERS: display start address: Must fall within the system area or machine language area (6000H ≦ display start address ≦ 7FFEH).

EXPLANATION:

1. Executing D only without specifying start and end addresses displays 8 bytes starting from the address following the last dumped address (initial value – 0).
2. Executing D [<display start address>] displays 8 bytes starting from the specified address.

EXAMPLE: Display of memory contents from memory address 7000H to 7008H.

[D][7][0][0][0][EXE]



```
)D7000
7000  48 45 4C 4C 4F 21 11 01
)_
```

E: EDIT MEMORY

PURPOSE: Changes the memory contents of the currently specified memory bank (RAM only).

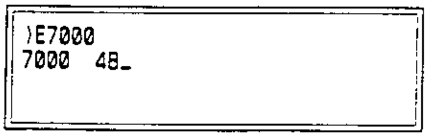
FORMAT: E [start address]

EXPLANATION:

1. Executing E only without specifying the start address allows editing of the address following the last address edited.
2. Values can be edited by using the 0 ~ 9 and A ~ F (including a ~ f) keys for input.
3. Pressing the [SPC] (space) key moves to the next address without changing the contents of the currently displayed address.
4. Pressing the [BS] (backspace) key returns to the previous address without changing the contents of the currently displayed address.
5. Pressing the [EXE] key exits the EDIT mode and returns to command input stand by.

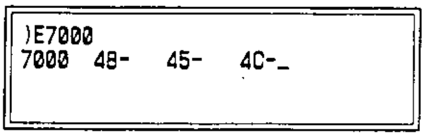
EXAMPLE: Changing the contents of address 7002H from 4C to 5A.

[E][7][0][0][0][EXE]



```
)E7000
7000  48_
```

[SPC] [SPC]



```
)E7000
7000  48-   45-   4C-_
```

APPENDICES

CHARACTER CODE TABLE

High-order 4 bits →

Low-order 4 bits ↓

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	(ROLL DOWN) 16	SPACE 32	Ø 48	@ 64	P 80	` 96	p 112	▁ 128	┴ 144	SPACE 160	ー 176	タ 192	ミ 208	═ 224	╳ 240
1	(ROLL UP) 1	(DEL) 17	! 33	1 49	A 65	Q 81	a 97	q 113	▂ 129	┬ 145	。 161	ア 177	チ 193	ム 209	╞ 225	円 241
2	(LINE TOP) 2	(INS) 18	" 34	2 50	B 66	R 82	b 98	r 114	▃ 130	┤ 146	「 162	イ 178	ツ 194	メ 210	╪ 226	年 242
3	3	19	# 35	3 51	C 67	S 83	c 99	s 115	▄ 131	├ 147	」 163	ウ 179	テ 195	モ 211	╡ 227	月 243
4	4	20	$ 36	4 52	D 68	T 84	d 100	t 116	▅ 132	▔ 148	、 164	エ 180	ト 196	ヤ 212	◢ 228	日 244
5	(LINE DEL) 5	21	% 37	5 53	E 69	U 85	e 101	u 117	▆ 133	─ 149	・ 165	オ 181	ナ 197	ユ 213	◣ 229	時 245
6	(LINE END) 6	22	& 38	6 54	F 70	V 86	f 102	v 118	▇ 134	│ 150	ヲ 166	カ 182	ニ 198	ヨ 214	◥ 230	分 246
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119	█ 135	▕ 151	ァ 167	キ 183	ヌ 199	ラ 215	◤ 231	秒 247
8	(BS) 8	(LINE C) 24	(40	8 56	H 72	X 88	h 104	x 120	▏ 136	┌ 152	ィ 168	ク 184	ネ 200	リ 216	♠ 232	〒 248
9	9	25	) 41	9 57	I 73	Y 89	i 105	y 121	▎ 137	┐ 153	ゥ 169	ケ 185	ノ 201	ル 217	♥ 233	市 249
A	10	26	* 42	: 58	J 74	Z 90	j 106	z 122	▍ 138	└ 154	ェ 170	コ 186	ハ 202	レ 218	◆ 234	区 250
B	(HOME) 11	27	+ 43	; 59	K 75	[91	k 107	{ 123	▌ 139	┘ 155	ォ 171	サ 187	ヒ 203	ロ 219	♣ 235	町 251
C	(CLS) 12	(→) 28	, 44	< 60	L 76	¥ 92	l 108	¦ 124	▋ 140	╭ 156	ャ 172	シ 188	フ 204	ワ 220	● 236	村 252
D	(CR LF) 13	(←) 29	– 45	= 61	M 77	] 93	m 109	} 125	▊ 141	╮ 157	ュ 173	ス 189	ヘ 205	ン 221	○ 237	人 253
E	14	(↑) 30	. 46	> 62	N 78	^ 94	n 110	~ 126	▉ 142	╰ 158	ョ 174	セ 190	ホ 206	゛ 222	╱ 238	▨ 254
F	15	(↓) 31	/ 47	? 63	O 79	_ 95	o 111	127	┼ 143	╯ 159	ッ 175	ソ 191	マ 207	゜ 223	╲ 239	255

- * Nothing is output for character codes for a character or function is not specified (indicated by a blank cell in the table).
- * Control codes are indicated by parentheses and are not displayed.
- * Characters which cannot be input directly can be displayed using the CHR$ function.
- * Because of display limitations, 88H, 89H and 8AH apear to be the same on the screen, but are different when printed out.
- * Character codes are hexadecimal values. 8AH should be represented as &H8A where &H indicates hexadecimal notation, 8 is the low-order 4 bits, and A is the high-order 4 bits.
- * Values in the lower right corner of each cell indicate the decimal value of the corresponding character code.

ERROR MESSAGE TABLE

Error code	Error message	Meaning	Correction
1	OM error	a) Insufficient memory or system over flow. b) Erroneous CLEAR statement specification.	a) Shorten program and check array dimensioning. b) Check CLEAR statement value. c) Use expansion RAM pack.
2	SN error	Erroneous command or statement format.	a) Check spelling of commands. b) Check program input.
3	ST error	String length exceeds 255 characters.	Shorten string to 255 characters or less.
4	TC error	Formula too complex.	Divide formula into smaller subformulas
5	BV error	a) I/O buffer overflow. b) Line length exceed 255 bytes or 256 characters.	a) Set RS-232C baud rate to lower value or set XON/OFF. b) Keep lines 255 characters or less in length.
6	NR error	I/O.device not ready for input/output.	a) Check connection and power switch of I/O device. b) Load a floppy disk into FDD.
7	RW error	a) Error generated in I/O device operation. b) LOAD of file for which FL error was generated at SAVE.	a) Check I/O device. b) Erase (kill) loaded program. NOTE: Repeated RW errors indicate that object program was not saved properly. Erase and resave, if possible.
8	BF error	Improper filename specification.	Check filename.
9	BN error	Improper file number specification.	Check file number.
10	NF error	Cannot find specified filename.	Recheck filename.
11	LB error	Low batteries in FDD.	a) Replace batteries in FDD. b) Use AC adaptor.
12	FL error	Disk space unavailable for writing.	a) Erase unneeded files. b) Use a new disk.
13	OV error	Value exceeds allowable calculation result or input range.	Check values.
14	MA error	a) Mathematical error such as division by zero. b) Argument exceeds allowable calculation range.	Check expressions and values.

Error code	Error message	Meaning	Correction
15	DD error	Double declaration of identical array.	Either erase previous array or use a different array name.
16	BS error	Subscript or parameter outside of allowable range.	a) Check subscripts. b) Increase size of arrays.
17	FC error	a) Erroneous use of function or statement. b) Illegal command used in direct mode or program mode. c) Illegal command used in CAL mode. d) Attempt to use undeclared array	a) Check argument values and statements. b) Check for statements that can not be used in respective mode. c) Check statements. d) Declare array using DIM statement.
18	UL error	a) Branch destination line number does not exist. b) Input of statement without line number in BASIC editing mode.	a) Check line numbers. b) Always use line numbers in BASIC editing mode. c) Enter BASIC programming mode.
19	TM error	a) Mismatch of variable type and contents. b) Mismatch of READ statement variable and data. c) Mismatch of INPUT# statement variable and data.	Check for illegal numeric assignment to string variables or string assignment to numeric variable.
20	RE error	RESUME statement outside of error handling routine.	Check RESUME statement location.
21	PR error	a) Attempt to write to password or write-protected disk or file. b) Execution of command that cannot be used with password protected files.	Cancel password or write protect status.
22	DA error	READ statement execution when no data present.	Check READ and DATA statements.
23	FO error	No FOR for NEXT statement.	Check for matching of FOR and NEXT statements.
24	NX error	No NEXT for FOR statement.	Check for matching of FOR and NEXT statements.
25	GS error	Mismatch of GOSUB and RETURN statements.	Check for matching of GOSUB and RETURN statements.
26	FM error	Unformatted or damaged disk.	Reformat disk or use new disk.

Error code	Error message	Meaning	Correction
27	FD error	FIELD statement length exceeds 256 characters.	Ensure data length specified by FIELD statement is 256 characters or less.
28	OP error	a) Attempt to access unopened file. b) Attempt to open already open file.	a) Execute OPEN statement. b) CLOSE file and then reopen.
29	AM error	a) Attempt to use random access for file opened for sequential access or vice versa. b) Attempt to use output-related command for device opened for input or vice versa. c) Attempt to load a random file. d) Attempt to use APPEND OPEN for BASIC or machine language file. e) Mismatched recorder baud rate. f) Attempt to execute machine language file without start address.	a) Do not use random access for sequential file and vice versa. b) Ensure proper used of input-related and output-related commands. c) Random files cannot be loaded. d) Do not use APPEND OPEN for BASIC files or machine language files. e) Check MT baud rate. f) Include start address.
30	FR error	Framing error detected by RS-232C port.	Check RS-232C connection and data transmission method.
31	PO error	Parity error or over run error detected by RS-232C port.	a) Check RS-232C connection and data transmission method. b) Use slower baud rate.
32	DF error	a) Undefined command sent to FDD. b) Abnormality in FDD.	a) Erroneous machine language program b) Disk contents may not be retained.
0	?? error	Undefined error.	Abnormal operation. Press RESET and check memory contents. If abnormal, press NEW ALL.

COMMAND/FUNCTION TABLE

COMMANDS

· CLEAR	©	· SYSTEM	©	· LIST	Ⓜ
· VARLIST	©	· EDIT	Ⓜ	· DELETE	Ⓜ
· RUN	Ⓜ	· TRON/TROFF	©	· END	
· STOP		· GOTO		· GOSUB/RETURN	
· ON GOTO		· ON GOSUB		· IF/THEN/ELSE	
· FOR/NEXT		· REM(')		· LET	©
· DATA/READ/RESTORE		· INPUT		· PRINT	©
· PRINT USING	©	· LOCATE	©	· ANGLE	©
· BEEP(ON/OFF)	©	· CLS	©	· DIM	©
· ERASE	©	· DRAW/DRAWC	©	· MON	©
· CALL	©	· ON ERROR GOTO		· RESUME	
· DEFCHR$	©	· PASS	Ⓜ	· NEW	Ⓜ
· STAT	©	· STAT CLEAR	©	· POKE	©

INPUT/OUTPUT COMMANDS

· LLIST	Ⓜ	· LPRINT	©	· LPRINT USING	©
· FORMAT	©	· BSAVE	©	· BLOAD	©
· OPEN		· CLOSE	©	· PRINT#	
· INPUT#		· SAVE	Ⓜ	· LOAD	Ⓜ
· PUT/GET		· FIELD		· RSET/LSET	
· VERIFY	©	· CHAIN	Ⓜ	· MERGE	Ⓜ
· LINEINPUT#		· PRINT# USING			

Ⓜ manual execution only ©manual or CAL mode execution

NOTE: Ⓜ and © are not included for commands which would be meaningless in manual execution.

SCIENTIFIC FUNCTIONS

· CHR $	· ASC	· STR $
· VAL	· MID $	· RIGHT $
· LFFT $	· LEN	· HEX $
· &H	· INKEY $	· INPUT $
· INPUT #	· DEG	· DMS $
· POINT		
· SIN	· COS	· TAN
· ASN	· ACS	· ATN
· HYP SIN	· HYP COS	· HYP TAN
· HYP ASN	· HYP ACS	· HYP ATN
· EXP	· LOG	· LGT
· SQR	· ABS	· SGN
· INT	· FRAC	· ROUND
· PI	· RND	· PEEK
· TAB	· FIX	
· CNT	· SUMX	· SUMY
· SUMXY	· SUMX2	· SUMY2
· MEANX	· MEANY	· SDX
· SDY	· SDXN	· SDYN
· LRA	· LRB	· COR
· EOX	· EOY	
· EOF	· ERR	· ERL
· LOF	· REV	· NORM
· TIME $	· DATE $	

RESERVED WORD LIST

A ABS
ACS
AND
ANGLE
APPEND
AS
ASC
ASN
ATN

B BEEP
BLOAD
BSAVE

C CALL
CHAIN
CHR$
CLEAR
CLOSE
CLS
CNT
COR
COS

D DATA
DATE$
DEF
DEG
DELETE
DIM
DMS$
DRAW
DRAWC

E EDIT
ELSE
END
EOF
EOX
EOY
ERASE
ERL
ERR
ERROR
EXP

F FIELD
FIX
FOR
FORMAT
FRAC

G GET
GOSUB
GOTO

H HEX$
HYP

I IF
INKEY$
INPUT
INT

L LEFT$
LEN
LET
LGT
LINE
LIST
LLIST
LOAD
LOCATE
LOF
LOG
LPRINT
LRA
LRB
LSET

M MEANX
MEANY
MERGE
MID$
MOD
MON

N NEW
NEXT
NORM
NOT

O OFF
ON
OPEN
OR
OUT

P PASS
PEEK
PI
POINT
POKE
PRINT
PUT

R READ
REM
RESUME
RESTORE
RETURN
REV
RIGHT$
RND
ROUND
RSET
RUN

S SAVE
SDX
SDXN
SDY
SDYN
SGN
SIN
SQR
STAT
STEP
STOP
STR$
SUMX
SUMX2
SUMXY
SUMY
SUMY2
SYSTEM

T TAB
TAN
THEN
TIME$
TO
TROFF
TRON

U USING

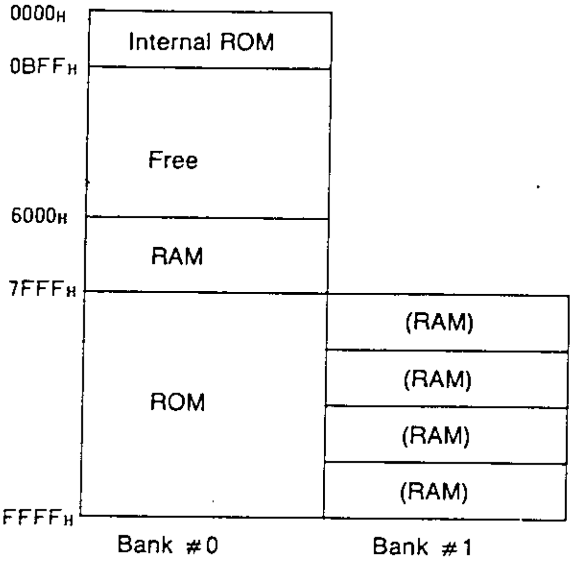
V VAL
VARLIST
VERIFY

X XOR

MEMORY MAP

Memory Free Areas

The HD61700 has free areas from addresses 00000H through 3FFFFH, (18 external addresses and 8 data addresses), while the free area of the computer is located at the addresses illustrated below.



The internal ROM indicated in the memory map is the HD61700 ROM (addresses 0H ~ BFFH, 3072 × 16 bits).

MNEMONIC TABLE

Mnemonic	Operation	Operation code								Hex	Byte	Machine cycle		State		Flags			
		7	6	5	4	3	2	1	0			Fetch	Execution	Fetch	Execution	Z	C	LZ	UZ
LD	LD $,$	0	0	0	0	0	0	1	0	02	3	2	2	6	6				
	LD $,($)	0	0	0	1	0	0	0	1	11	3	2	4	6	11				
	LD $,(IX± $)	0	0	1	0	1	0	0	0	28	3	2	4	6	11				
	LD $,(IZ± $)	0	0	1	0	1	0	0	1	29	3	2	4	6	11				
	LD $,n	0	1	0	0	0	0	1	0	42	3	2	2	6	6				
	LD $,(IX±n)	0	1	1	0	1	0	0	0	68	3	2	4	6	11				
	LD $,(IZ±n)	0	1	1	0	1	0	0	1	69	3	2	4	6	11				
LDI	LDI $,(IX± $)	0	0	1	0	1	0	1	0	2A	3	2	4	6	11				
	LDI $,(IZ± $)	0	0	1	0	1	0	1	1	2B	3	2	4	6	11				
	LDI $,(IX±n)	0	1	1	0	1	0	1	0	6A	3	2	4	6	11				
	LDI $,(IZ±n)	0	1	1	0	1	0	1	1	6B	3	2	4	6	11				
ST	ST $,($)	0	0	0	1	0	0	0	0	10	3	2	4	6	11				
	ST $,(IX± $)	0	0	1	0	0	0	0	0	20	3	2	4	6	11				
	ST $,(IZ± $)	0	0	1	0	0	0	0	1	21	3	2	4	6	11				
	ST $,(IX±n)	0	1	1	0	0	0	0	0	60	3	2	4	6	11				
	ST $,(IZ±n)	0	1	1	0	0	0	0	1	61	3	2	4	6	11				
STI	STI $,(IX± $)	0	0	1	0	0	0	1	0	22	3	2	4	6	11				
	STI $,(IZ± $)	0	0	1	0	0	0	1	1	23	3	2	4	6	11				
	STI $,(IX±n)	0	1	1	0	0	0	1	0	62	3	2	4	6	11				
	STI $,(IZ±n)	0	1	1	0	0	0	1	1	63	3	2	4	6	11				
PHS	PHS $	0	0	1	0	0	1	1	0	26	2	1	3	3	9				
PHU	PHU $	0	0	1	0	0	1	1	1	27	2	1	3	3	9				
PPS	PPS $	0	0	1	0	1	1	1	0	2E	2	1	4	3	11				
PPU	PPU $	0	0	1	0	1	1	1	1	2F	2	1	4	3	11				
GFL	GFL $	0	0	0	1	1	1	0	0	1C	2	1	2	3	6				
GPO	GPO $	0	0	0	1	1	1	0	0	1C	2	1	2	3	6				
GST	GST PE; $	0	0	0	1	1	1	1	0	1E	2	1	2	3	6				
	GST PD, $	0	0	0	1	1	1	1	0	1E	2	1	2	3	6				
	GST UA, $	0	0	0	1	1	1	1	0	1E	2	1	2	3	6				
	GST IA, $	0	0	0	1	1	1	1	1	1F	2	1	2	3	6				
	GST IE, $	0	0	0	1	1	1	1	1	1F	2	1	2	3	6				
	GST TM, $	0	0	0	1	1	1	1	1	1F	2	1	2	3	6				
PFL	PFL $	0	0	0	1	0	1	0	0	14	2	1	2	3	6	M	M	M	M
PST	PST PE, $	0	0	0	1	0	1	1	0	16	2	1	2	3	6				
	PST PD, $	0	0	0	1	0	1	1	0	16	2	1	2	3	6				
	PST UA, $	0	0	0	1	0	1	1	0	16	2	1	2	3	6				
	PST IA, $	0	0	0	1	0	1	1	1	17	2	1	2	3	6				
	PST IE, $	0	0	0	1	0	1	1	1	17	2	1	2	3	6				
	PST PE,n	0	1	0	1	0	1	1	0	56	3	2	2	6	6				
	PST PD,n	0	1	0	1	0	1	1	0	56	3	2	2	6	6				
	PST UA,n	0	1	0	1	0	1	1	0	56	3	2	2	6	6				
	PST IA,n	0	1	0	1	0	1	1	1	57	3	2	2	6	6				
	PST IE,n	0	1	0	1	0	1	1	1	57	3	2	2	6	6				
LDW	LDW $,$	1	0	0	0	0	0	1	0	82	3	2	4	6	11				

Mnemonic	Operation	Operation code								Hex	Byte	Machine cycle		State		Flags			
		7	6	5	4	3	2	1	0			Fetch	Execution	Fetch	Execution	Z	C	LZ	UZ
	LDW $,($)	1	0	0	1	0	0	0	1	91	3	2	5	6	14				
	LDW $,(IX± $)	1	0	1	0	1	0	0	0	A8	3	2	5	6	14				
	LDW $,(IZ± $)	1	0	1	0	1	0	0	1	A9	3	2	5	6	14				
	LDW $,m	1	1	0	1	0	0	0	1	D1	4	3	5	9	14				
LDIW	LDIW $,(IX± $)	1	0	1	0	1	0	1	0	AA	3	2	5	6	14				
	LDIW $,(IZ± $)	1	0	1	0	1	0	1	1	AB	3	2	5	6	14				
STW	STW $,(IX± $)	1	0	1	0	0	0	0	0	A0	3	2	5	6	14				
	STW $,(IZ± $)	1	0	1	0	0	0	0	1	A1	3	2	5	6	14				
	STW $,($)	1	0	0	1	0	0	0	0	90	3	2	5	6	14				
STIW	STIW $,(IX± $)	1	0	1	0	0	0	1	0	A2	3	2	4	6	14				
	STIW $,(IZ± $)	1	0	1	0	0	0	1	1	A3	3	2	4	6	14				
PHSW	PHSW $	1	0	1	0	0	1	1	0	A6	2	1	4	3	12				
PHUW	PHUW $	1	0	1	0	0	1	1	1	A7	2	1	4	3	12				
PPSW	PPSW $	1	0	1	0	1	1	1	0	AE	2	1	5	3	14				
PPUW	PPUW $	1	0	1	0	1	1	1	1	AF	2	1	5	3	14				
GRE	GRE IX,$	1	0	0	1	1	1	1	0	9E	2	1	4	3	11				
	GRE IY,$	1	0	0	1	1	1	1	0	9E	2	1	4	3	11				
	GRE IZ,$	1	0	0	1	1	1	1	0	9E	2	1	4	3	11				
	GRE US,$	1	0	0	1	1	1	1	0	9E	2	1	4	3	11				
	GRE SS,$	1	0	0	1	1	1	1	1	9F	2	1	4	3	11				
	GRE KY, $	1	0	0	1	1	1	1	1	9F	2	1	4	3	11				
PRE	PRE IX,$	1	0	0	1	0	1	1	0	96	2	1	4	3	11				
	PRE IY,$	1	0	0	1	0	1	1	0	96	2	1	4	3	11				
	PRE IZ,$	1	0	0	1	0	1	1	0	96	2	1	4	3	11				
	PRE US,$	1	0	0	1	0	1	1	0	96	2	1	4	3	11				
	PRE SS,$	1	0	0	1	0	1	1	1	97	2	1	4	3	11				
	PRE IX,m	1	1	0	1	0	1	1	0	D6	4	3	4	9	11				
	PRE IY,m	1	1	0	1	0	1	1	0	D6	4	3	4	9	11				
	PRE IZ,m	1	1	0	1	0	1	1	0	D6	4	3	4	9	11				
	PRE US,m	1	1	0	1	0	1	1	0	D6	4	3	4	9	11				
	PRE SS,m	1	1	0	1	0	1	1	1	D7	4	3	4	9	11				
AD	AD $,$	0	0	0	0	1	0	0	0	08	3	2	2	6	6	M	M	M	M
	AD (IX± $),$	0	0	1	1	1	1	0	0	3C	3	2	4	6	12	M	M	M	M
	AD (IZ± $),$	0	0	1	1	1	1	0	1	3D	3	2	4	6	12	M	M	M	M
	AD $,n	0	1	0	0	1	0	0	0	48	3	2	2	6	6	M	M	M	M
	AD (IX±n),$	0	1	1	1	1	1	0	0	7C	3	2	4	6	12	M	M	M	M
	AD (IZ±n),$	0	1	1	1	1	1	0	1	7D	3	2	4	6	12	M	M	M	M
ADB	ADB $,$	0	0	0	0	1	0	1	0	0A	3	2	2	6	6	M	M	M	M
	ADB $,n	0	1	0	0	1	0	1	0	4A	3	2	2	6	6	M	M	M	M
ADC	ADC $,$	0	0	0	0	0	0	0	0	00	3	2	2	6	6	M	M	M	M
	ADC (IX± $),$	0	0	1	1	1	0	0	0	38	3	2	4	6	12	M	M	M	M
	ADC (IZ± $),$	0	0	1	1	1	0	0	1	39	3	2	4	6	12	M	M	M	M
	ADC $,n	0	1	0	0	0	0	0	0	40	3	2	2	6	6	M	M	M	M
	ADC (IX±n),$	0	1	1	1	1	0	0	0	78	3	2	4	6	12	M	M	M	M

Mnemonic	Operation	Operation code								Hex	Byte	Machine cycle		State		Flags			
		7	6	5	4	3	2	1	0			Fetch	Execution	Fetch	Execution	Z	C	LZ	UZ
	ADC (IZ±n),$	0	1	1	1	1	0	0	1	79	3	2	4	6	12	M	M	M	M
AN	AN $,$	0	0	0	0	1	1	0	0	OC	3	2	2	6	6	M	0	M	M
	AN $,n	0	1	0	0	1	1	0	0	4C	3	2	2	6	6	M	0	M	M
ANC	ANC $,$	0	0	0	0	0	1	0	0	04	3	2	2	6	6	M	0	M	M
	ANC $,n	0	1	0	0	0	1	0	0	44	3	2	2	6	6	M	0	M	M
NA	NA $,$	0	0	0	0	1	1	0	1	0D	3	2	2	6	6	M	1	M	M
	NA $,n	0	1	0	0	1	1	0	1	4D	3	2	2	6	6	M	1	M	M
NAC	NAC $,$	0	0	0	0	0	1	0	1	05	3	2	2	6	6	M	1	M	M
	NAC $,n	0	1	0	0	0	1	0	1	45	3	2	2	6	6	M	1	M	M
OR	OR $,$	0	0	0	0	1	1	1	0	0E	3	2	2	6	6	M	1	M	M
	OR $,n	0	1	0	0	1	1	1	0	4E	3	2	2	6	6	M	1	M	M
ORC	ORC $,$	0	0	0	0	0	1	1	0	06	3	2	2	6	6	M	1	M	M
	ORC $,n	0	1	0	0	0	1	1	0	46	3	2	2	6	6	M	1	M	M
SB	SB $,$	0	0	0	0	1	0	0	1	09	3	2	2	6	6	M	M	M	M
	SB (IX± $),$	0	0	1	1	1	1	1	0	3E	3	2	4	6	12	M	M	M	M
	SB (IZ± $),$	0	0	1	1	1	1	1	1	3F	3	2	4	6	12	M	M	M	M
	SB $,n	0	1	0	0	1	0	0	1	49	3	2	2	6	6	M	M	M	M
	SB (IX±n),$	0	1	1	1	1	1	1	0	7E	3	2	4	6	12	M	M	M	M
	SB (IZ±n),$	0	1	1	1	1	1	1	1	7F	3	2	4	6	12	M	M	M	M
SBB	SBB $,$	0	0	0	0	1	0	1	1	0B	3	2	2	6	6	M	M	M	M
	SBB $,n	0	1	0	0	1	0	1	1	4B	3	2	2	6	6	M	M	M	M
SBC	SBC $,$	0	0	0	0	0	0	0	1	01	3	2	2	6	6	M	M	M	M
	SBC (IX± $),$	0	0	1	1	1	1	1	0	3E	3	2	4	6	12	M	M	M	M
	SBC (IZ± $),$	0	0	1	1	1	1	1	1	3F	3	2	4	6	12	M	M	M	M
	SBC $,n	0	1	0	0	0	0	0	1	41	3	2	2	6	6	M	M	M	M
	SBC (IX±n),$	0	1	1	1	1	0	1	0	7A	3	2	4	6	12	M	M	M	M
	SBC (IZ±n),$	0	1	1	1	1	0	1	1	7B	3	2	4	6	12	M	M	M	M
XR	XR $,$	0	0	0	0	1	1	1	1	0F	3	2	2	6	6	M	0	M	M
	XR $,n	0	1	0	0	1	1	1	1	4F	3	2	2	6	6	M	0	M	M
XRC	XRC $,$	0	0	0	0	0	1	1	1	07	3	2	2	6	6	M	0	M	M
	XRC $,n	0	1	0	0	0	1	1	1	47	3	2	2	6	6	M	0	M	M
ADW	ADW $,$	1	0	0	0	1	0	0	0	88	3	2	4	6	11	M	M	M	M
	ADW (IX± $),$	1	0	1	1	1	1	0	0	BC	3	2	6	6	18	M	M	M	M
	ADW (IZ± $),$	1	0	1	1	1	1	0	1	BD	3	2	6	6	18	M	M	M	M
ADBW	ADBW $,$	1	0	0	0	1	0	1	0	8A	3	2	4	6	11	M	M	M	M
ADCW	ADCW $,$	1	0	0	0	0	0	0	0	80	3	2	4	6	11	M	M	M	M
	ADCW (IX± $),$	1	0	1	1	1	0	0	0	B8	3	2	4	6	11	M	M	M	M
	ADCW (IZ± $),$	1	0	1	1	1	0	0	1	B9	3	2	4	6	11	M	M	M	M
ANW	ANW $,$	1	0	0	0	1	1	0	0	8C	3	2	4	6	11	M	0	M	M
ANCW	ANCW $,$	1	0	0	0	0	1	0	0	84	3	2	4	6	11	M	0	M	M
NAW	NAW $,$	1	0	0	0	1	1	0	1	8D	3	2	4	6	11	M	1	M	M
NACW	NACW $,$	1	0	0	0	0	1	0	1	85	3	2	4	6	11	M	1	M	M
ORW	ORW $,$	1	0	0	0	1	1	1	0	8E	3	2	4	6	11	M	1	M	M
ORCW	ORCW $,$	1	0	0	0	0	1	1	0	86	3	2	4	6	11	M	1	M	M

Mnemonic	Operation	Operation code								Hex	Byte	Machine cycle		State		Flags			
		7	6	5	4	3	2	1	0			Fetch	Execution	Fetch	Execution	Z	C	LZ	UZ
SBW	SBW $,$	1	0	0	0	1	0	0	1	8 9	3	2	4	6	1 1	M	M	M	M
	SBW (IX± $), $	1	0	1	1	1	1	1	0	B E	3	2	6	6	1 8	M	M	M	M
	SBW (IZ± $),$	1	0	1	1	1	1	1	1	B F	3	2	6	6	1 8	M	M	M	M
SBBW	SBBW $,$	1	0	0	0	1	0	1	1	8 B	3	2	6	6	1 8	M	M	M	M
SBCW	SBCW $,$	1	0	0	0	0	0	0	1	8 1	3	2	4	6	1 1	M	M	M	M
	SBCW (IX± $),$	1	0	1	1	1	0	1	0	B A	3	2	6	6	1 8	M	M	M	M
	SBCW (IZ± $),$	1	0	1	1	1	0	1	1	B B	3	2	6	6	1 8	M	M	M	M
XRW	XRW $,$	1	0	0	0	1	1	1	1	8 F	3	2	4	6	1 1	M	0	M	M
XRCW	XRCW $,$	1	0	0	0	0	1	1	1	8 7	3	2	4	6	1 1	M	0	M	M
BID	BID $	0	0	0	1	1	0	0	0	1 8	2	1	2	3	6	M	M	M	M
BIU	BIU $	0	0	0	1	1	0	0	0	1 8	2	1	2	3	6	M	M	M	M
ROD	ROD $	0	0	0	1	1	0	0	0	1 8	2	1	2	3	6	M	M	M	M
ROU	ROU $	0	0	0	1	1	0	0	0	1 8	2	1	2	3	6	M	M	M	M
DID	DID $	0	0	0	1	1	0	1	0	1 A	2	1	2	3	6	M	0	M	0
DIU	DIU $	0	0	0	1	1	0	1	0	1 A	2	1	2	3	6	M	0	0	M
CMP	CMP $	0	0	0	1	1	0	1	1	1 B	2	1	2	3	6	M	M	M	M
INV	INV $	0	0	0	1	1	0	1	1	1 B	2	1	2	3	6	M	1	M	M
BIDW	BIDW $	1	0	0	1	1	0	0	0	9 8	2	1	4	3	1 1	M	M	M	M
BIUW	BIUW $	1	0	0	1	1	0	0	0	9 8	2	1	4	3	1 1	M	M	M	M
RODW	RODW $	1	0	0	1	1	0	0	0	9 8	2	1	4	3	1 1	M	M	M	M
ROUW	ROUW $	1	0	0	1	1	0	0	0	9 8	2	1	4	3	1 1	M	M	M	M
DIDW	DIDW $	1	0	0	1	1	0	1	0	9 A	2	1	4	3	1 1	M	0	M	M
DIUW	DIUW $	1	0	0	1	1	0	1	0	9 A	2	1	4	3	1 1	M	0	M	M
BYDW	BYDW $	1	0	0	1	1	0	1	0	9 A	2	1	4	3	1 1	M	0	M	M
BYUW	BYUW $	1	0	0	1	1	0	1	0	9 A	2	1	4	3	1 1	M	0	M	M
CMPW	CMPW $	1	0	0	1	1	0	1	1	9 B	2	1	4	3	1 1	M	M	M	M
INVW	INVW $	1	0	0	1	1	0	1	1	9 B	2	1	4	3	1 1	M	1	M	M
JP	JP Z,m	0	0	1	1	0	0	0	0	3 0	3	2	2	6	6				
	JPNC,m	0	0	1	1	0	0	0	1	3 1	3	2	2	6	6				
	JP LZ,m	0	0	1	1	0	0	1	0	3 2	3	2	2	6	6				
	JP UZ,m	0	0	1	1	0	0	1	1	3 3	3	2	2	6	6				
	JP NZ,m	0	0	1	1	0	1	0	0	3 4	3	2	2	6	6				
	JP C,m	0	0	1	1	0	1	0	1	3 5	3	2	2	6	6				
	JP m	0	0	1	1	0	1	1	1	3 7	3	2	2	6	6				
JR	JR Z,±P	1	0	1	1	0	0	0	0	B 0	2	1	2	3	6				
	JR NC,±P	1	0	1	1	0	0	0	1	B 1	2	1	2	3	6				
	JR LZ,±P	1	0	1	1	0	0	1	0	B 2	2	1	2	3	6				
	JR UZ,±P	1	0	1	1	0	0	1	1	B 3	2	1	2	3	6				
	JR NZ,±P	1	0	1	1	0	1	0	0	B 4	2	1	2	3	6				
	JR C,±P	1	0	1	1	0	1	0	1	B 5	2	1	2	3	6				
	JR ±P	1	0	1	1	0	1	1	1	B 7	2	1	2	3	6				
CAL	CAL Z,m	0	1	1	1	0	0	0	0	7 0	3	2	4	6	1 2				
	CAL NC,m	0	1	1	1	0	0	0	1	7 1	3	2	4	6	1 2				
	CAL LZ,m	0	1	1	1	0	0	1	0	7 2	3	2	4	6	1 2				

Mnemonic	Operation	Operation code								Hex	Byte			State		Flags			
		7	6	5	4	3	2	1	0			Fetch	Execution	Fetch	Execution	Z	C	LZ	UZ
	CAL UZ,m	0	1	1	1	0	0	1	1	7 3	3	2	4	6	1 2				
	CAL NZ,m	0	1	1	1	0	1	0	0	7 4	3	2	4	6	1 2				
	CAL C,m	0	1	1	1	0	1	0	1	7 5	3	2	4	6	1 2				
	CAL m	0	1	1	1	0	1	1	1	7 7	3	2	4	6	1 2				
RTN	RTN Z	1	1	1	1	0	0	0	0	F 0	1	0	5	0	1 4				
	RTN NC	1	1	1	1	0	0	0	1	F 1	1	0	5	0	1 4				
	RTN LZ	1	1	1	1	0	0	1	0	F 2	1	0	5	0	1 4				
	RTN UZ	1	1	1	1	0	0	1	1	F 3	1	0	5	0	1 4				
	RTN NZ	1	1	1	1	0	1	0	0	F 4	1	0	5	0	1 4				
	RTN C	1	1	1	1	0	1	0	1	F 5	1	0	5	0	1 4				
	RTN	1	1	1	1	0	1	1	1	F 7	1	0	5	0	1 4				
BDN	BDN	1	1	0	1	1	0	0	1	D 9	1	0	2a+2	0	6a+6				
BUP	BUP	1	1	0	1	1	0	0	0	D 8	1	0	2a+2	0	6a+6				
SDN	SDN $	1	1	0	1	1	1	0	1	D D	2	1	2a+2	3	6a+6	M	M	M	M
	SDN n	0	1	0	1	1	1	0	1	5 D	2	1	2a+2	3	6a+6	M	M	M	M
SUP	SUP $	1	1	0	1	1	1	0	0	D C	2	1	2a+2	3	6a+6	M	M	M	M
	SUP n	0	1	0	1	1	1	0	0	5 C	2	1	2a+2	3	6a+6	M	M	M	M
NOP	NOP	1	1	1	1	1	0	0	0	F 8	1	0	2	0	6				
CLT	CLT	1	1	1	1	1	0	0	1	F 9	1	0	2	0	6				
FST	FST	1	1	1	1	1	0	1	0	F A	1	0	2	0	6				
SLW	SLW	1	1	1	1	1	0	1	1	F B	1	0	2	0	6				
CANI	CANI	1	1	1	1	1	1	0	0	F C	1	0	2	0	6				
RTNI	RTNI	1	1	1	1	1	1	0	1	F D	1	0	5	0	1 4				
OFF	OFF	1	1	1	1	1	1	1	0	F E	1	0	2	0	6				
TRP	TRP	1	1	1	1	1	1	1	1	F F	1	0	4	0	1 2				

n : 8-bit immediate data
m: 16-bit immediate data
p : 7-bit immediate data
a : Number of bytes processed.

SECONDARY OPERATION COMMANDS

Mnemonic	Operation	Second operation code							
		7	6	5	4	3	2	1	0
GFL			1	0					
GPO			0	0					
GST	GST PE		0	0					
	PD		0	1					
	UA		1	1					
	IA		0	0					
	IE		0	1					
	TM		1	1					
PST	PST PE		0	0					
	PD		0	1					
	UA		1	1					
	IA		0	0					
	IE		0	1					
GRE	GRE IX		0	0					
	IY		0	1					
	IZ		1	0					
	US		1	1					
	SS		0	0					
	KY		1	1					
PRE	PRE IX		0	0					
	IY		0	1					
	IZ		1	0					
	US		1	1					
	SS		0	0					
BID			1	0					
BIU			1	1					
ROD			0	0					
ROU			0	1					
DID			0	0					
DIU			0	1					
CMP			0	0					
INV			1	0					
BIDW			1	0					
BIUW			1	1					
RODW			0	0					
ROUW			0	1					
DIDW			0	0					
DIUW			0	1					
BYDW			1	0					
BYUW			1	1					
CMPW			0	0					
INVW			1	0					

INDEX

BASIC

A

&H ... 83
ABS ... 70
ANGLE ... 63
ASC ... 76
ASN/ACS/ATN ... 65

B

BEEP ... 50
BLOAD ... 110
BSAVE ... 109

C

CALL ... 56
CHAIN ... 113
CHR$ ... 76
CLEAR ... 22
CLOSE ... 97
CLS ... 48
CNT ... 88
COR ... 91
COS ... 64

D

DATA ... 40
DATE$ ... 86
DEFCHR$ ... 49
DEG ... 84
DELETE ... 24
DIM ... 54
DMS$ ... 84
DRAW/DRAWC ... 58

E

EDIT ... 26
END ... 29
EOF ... 102
EOX ... 92
EOY ... 92
ERASE ... 55
ERL ... 62
ERR ... 62
EXP ... 68

F

FIELD ... 103
FIX ... 73
FORMAT ... 108
FOR ~ TO ~ STEP ~ NEXT ... 37
FRAC ... 73

G

GET ... 106
GOSUB ... 32
GOTO ... 31

H

HEX$ ... 82
HYPASN/HYPACS/HYPATN ... 67
HYPSIN/HYPCOS/HYPTAN ... 66

I

IF ~ THEN ~ ELSE/IF ~ GOTO ~ ELSE ... 36
INKEY$ ... 52
INPUT ... 51
INPUT# ... 100
INPUT$ ... 53
INT ... 72

L

LEFT$ ... 81
LEN ... 82
LET ... 39
LINE INPUT# ... 101
LIST ... 25
LLIST ... 93
LOAD ... 112
LOCATE ... 47
LOF ... 107
LOG/LGT ... 69
LPRINT ... 94
LPRINT USING ... 95
LRA ... 91
LRB ... 91
LSET ... 105

M

MEANX........................90
MEANY........................90
MERGE........................114
MID$.........................79
MON..........................24

N

NEW..........................21
NORM.........................45

O

ON ERROR GOTO................60
ON ~ GOSUB...................35
ON ~ GOTO....................34
OPEN.........................96

P

PASS.........................20
PEEK.........................56
PI...........................74
POINT........................59
POKE.........................57
PRINT........................43
PRINT USING..................46
PRINT#.......................98
PRINT# USING.................99
PUT..........................106

R

READ.........................41
REM(')........................38
RESTORE......................42
RESUME.......................61
RETURN.......................33
REV..........................45
RIGHT$.......................80
RND..........................75
ROUND........................74
RSET.........................104
RUN..........................27

S

SAVE.........................111
SDX..........................90
SDXN.........................90
SDY..........................90
SDYN.........................90
SGN..........................71
SIN..........................64
SQR..........................70
STAT.........................87
STAT CLEAR...................88
STOP.........................30
STR$.........................77
SUMX.........................89
SUMX2........................89
SUMXY........................89
SUMY.........................89
SUMY2........................89
SYSTEM.......................21

T

TAB..........................44
TAN..........................64
TIME$........................85
TROFF........................28
TRON.........................28

V

VAL..........................78
VARLIST......................26
VERIFY.......................114

ASSEMBLER

A

AD 138
ADB 141
ADBW......................... 154
ADC 142
ADCW 155
ADW.......................... 152
AN 145
ANC 148
ANCW 159
ANW.......................... 157

B

BDN 180
BID.......................... 162
BIDW 166
BIU.......................... 162
BIUW 166
BUP 179
BYDW......................... 169
BYUW 168

C

CAL 175
CANI......................... 185
CLT.......................... 184
CMP 164
CMPW 170

D

DID.......................... 163
DIDW 168
DIU.......................... 163
DIUW 167

F

FST.......................... 184

G

GFL 130
GPO.......................... 130
GRE.......................... 136
GST 131

I

INV.......................... 164
INVW 169

J

JP........................... 171
JR........................... 173

L

LDI.......................... 127
LDIW 133
LDW.......................... 132

N

NA 146
NAC 149
NACW 159
NAW.......................... 157
NOP 184

O

OFF 184
OR........................... 146
ORC.......................... 149
ORCW 159
ORW.......................... 158

P

PFL.......................... 130
PHS 129
PHSW......................... 136
PHU 130
PHUW 136
PPS 129
PPSW......................... 135
PPU 129
PPUW......................... 135
PRE 137
PST 131

R

ROD 161
RODW 165
ROU 161
ROUW 165
RTN 177
RTNI 185

S

SB 140
SBB 142
SBBW 154
SBC 144
SBCW 156
SBW 153
SDN 183
SLW 184
ST 128
STI 128
STIW 135
STW 134
SUP 182

T

TRP 185

X

XR 147
XRC 150
XRCW 160
XRW 158