# PRELIMINARY DIP INTERPRETIVE PROGRAM FOR THE DDP-24 GENERAL PURPOSE COMPUTER

COMPUTER CONTROL COMPANY

# CONTENTS

# GLOSSARY

DIP is the name given to a simplified programming system for Computer Control Company's DDP-24. It is an interpretive system, employing features of DDP-24, intended for the occasional or neophyte user.

A COMMAND is an instruction to DIP to perform a specific operation on data located at a specific place within the computer. A list of DIP commands is included in this manual.

DATA is a group of decimal digits which represents a numerical value occupying a location in the computer memory. Data may be read into DIP in fixed or floating point notation.

A PROGRAM is a set of commands which when obeyed by DIP extracts some desired result from data.

AN ADDRESS is a decimal integer specifying some location in the computer memory.

A FIXED POINT NUMBER is a decimal numerical value expressed in ordinary notation, such as "-357.62" or "+.783". A sign and decimal point may or may not be included.

A FLOATING POINT NUMBER is a decimal numeric value expressed in scientific notation, such as "0.35762E+3" or "+.783EO". The signs and decimal point may or may not be included. The portion preceding the E is always considered fractional; the number following the E signifies what power of 10 the fraction is to be multiplied by. For example, 3.14 expressed as a floating point number would be: +.314E+1, or for brevity, it may be written 314El. Similarly, -.000123 would be written -.123E-3.

# SECTION I

# INTRODUCTION

In the past, the beginning computer user has been forced to spend many hours learning about computers in general and his computer in particular before he could actually program the solution of any problem. The occasional computer user has spent precious time searching voluminous manuals for some remote or forgotten fact which he needs to complete a program.

Knowledge of the internal workings of computers and such functions as input-output, number systems, assemblers, compilers, and their rules and exceptions is important and necessary only to those who use this knowledge daily.

The DIP system has been designed to fill this gap in computer user capacity. DIP language is easily learned and is intended for the occasional user or the user new to the field. It is a simplified computer language and has a minimum of rules. DIP is most helpful in programming the "one time" problem and as an educational tool to aid in learning the basic features of the DDP-24. While DIP does not answer the professional programmer's need for many detailed functions, it does provide a comprehensive and powerful set of commands for problem solution.

In essence, DIP is an interpreter for the DDP-24. That is, it is a service program in machine language which, when loaded into the DDP-24 computer, will aid the computer and the user in executing programs written in simple DIP language. DIP is actually a programmed simulator of a computer. It has registers, a memory, and the capacity to do arithmetic and compare numbers just as any other computer. However, this "simulated computer" provides a link between the DDP-24 and the user. It presents the user with a greatly simplified version of the DDP-24 which the user can easily command and control. The DIP interpreter program requires only a portion of the computer memory for its storage. The remainder of the memory is available to the user to store data and DIP commands.

DIP language is a set of commands written in specified format. The simulated computer or interpreter understands these commands and executes them. These commands are general in nature and would usually require the execution of many DDP-24 instructions to duplicate the function of one DIP command.

The system is designed to aid in the solution of mathematical problems. Therefore, it is capable of working with numerical data only. Data may be presented to the computer as decimal numbers in fixed or floating point notation.

DIP provides its user with nearly all the powerful features of the DDP-24 computer. These include: data movement, arithmetic manipulation, command sequence control, index registers, indirect addressing, and input-output. Many commonly used transcendental functions are a part of the command repertoire. All numerical values used in the system are decimal. In addition, the interpreter provides powerful functions to aid in finding and correcting program errors.

The system can be operated in either of two ways. It can act as a stored program computer, executing commands stored in memory automatically, or in a mode similar to a desk calculator, executing commands as they are entered without storage.

# SECTION II
# DIP CHARACTERISTICS

## THE DIP INTERPRETER

The DIP interpreter is a program simulated, simple computer existing within the confines of the DDP-24. Some additional features, in the form of debugging and mathematical problem solution commands, extend its capacity beyond that of the DDP-24 instruction repertoire. Some restrictions in flexibility make these extensions possible. The interpreter is most useful in the solution of the "one-time" mathematical problem and as an educational tool for learning the features of the DDP-24 and programming in general.

The internal components of the interpreter include a memory, an accumulator, two limit registers, ten index registers, an indirect addressing capability, two operating modes, and a comprehensive command repertoire. A powerful set of debugging aids is also included.

## MEMORY

Only a small portion of the DDP-24 computer memory is occupied by the interpreter program itself. The remainder of the memory is available to the programmer to store DIP commands and/or data words. This available memory is read into and written from through the use of the input-output commands described in the command repertoire.

Data words and commands are thought of as occupying some location within the available memory. This location is defined by a decimal integer called an "address." All available memory is divided into fixed length locations which can be referenced by their address. Once a data word or command is stored at some address, the information will remain in that address until new information replaces it. The locations in available memory have addresses beginning with zero and continuing sequentially to a maximum value dependent on the size of the DDP-24 memory.

## ACCUMULATOR

The accumulator is a special register in the interpreter set aside to receive the results of arithmetic and transcendental operations. It also acts as an intermediate storage point for the movement of data words and commands. The contents of the accumulator can be tested for positive or negative sign and zero magnitude through the applicable jump commands. The contents of the accumulator can be displayed by executing a store command to some memory location and displaying that memory location.

The contents of the accumulator remain undisturbed except during the actual load and arithmetic commands which intentionally modify these contents. Whenever the accumulator is loaded or filled with a result, the previous contents of the accumulator are lost.

## LIMIT REGISTERS

Two limit registers are provided in the interpreter to aid in specifying input and output commands. These are called the input limit register and the output limit register. The function of these registers is to contain a limiting integer specifying the number of data words or commands to be read into or written out of the computer. The input limit register is also used in the CLEAR MEMORY command to specify the number of locations to be cleared.

The limit registers have no sign associated with them and they are always assumed to contain positive integers. The values contained in the registers remain unchanged until the registers are reloaded. The limit register can be used any number of times within any program. A value of zero in a limit register is not equivalent to a value of one. A zero value defeats the function of the command which uses the value in the limit register.

## INDEX REGISTERS

An index register is a special register in which any number up to the largest possible address can be stored. Index registers provide a convenient means of tallying the number of times a command or group of commands is executed or of allowing the computer to obey the same group of commands a number of times using data from a different set of addresses each time.

Each address modified by an index register specifies an effective address which is the sum of the address and the contents of the index register. Neither the address nor the index register have a sign, so the summation is always positive. The summation is made only at execution time in a special register; the address portion of the command in memory is left unchanged.

The same index register may be used any number of times with any number of commands. The value in the index register remains unchanged except after a JUMP ON INDEX INCREMENTED is executed. DIP provides ten index registers numbered 0-9 so groups of commands may have their addresses modified in different ways. The desired index register is specified in the X portion of the command following the address and separated from the address by a comma.

### Example No. 1

Consider that index register 3 contains a value of 9 and the command, load the accumulator from location 65, is executed modified by index register 3.

```
            LDA            65, 3
```

The computer will decode the command in a special register and add 9 to 65 to establish the modified or effective address 74 and will load the contents of location 74 into the accumulator.

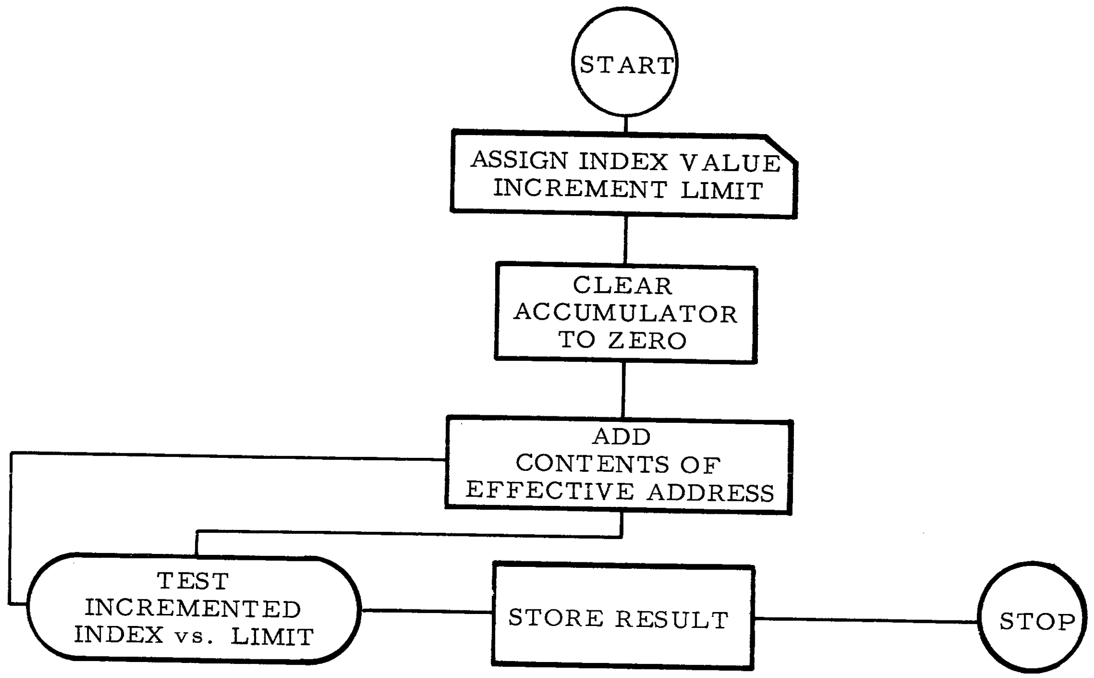
By changing the value of index register 3, the command in the example above can be made to load the contents of any location into the accumulator.

Each index register has two other associated pieces of information which are very useful. These are the "increment" and the "limit." The "increment" specifies the amount to change the register when it is incremented. The increment is always a positive integer. The "limit" specifies the value at which incrementation is to stop and an alternate program path is to be taken. Separate commands for loading, incrementing, and testing limits are provided in the command repertoire.

Example No. 2

A program is written below to find the sum of the data elements in locations 40 through 69. This sum will be stored in location 100. Notice that index register 2 will cause the command in location 14 to add to the accumulator in turn the contents of locations 40 through 69. This program is run in automatic mode (beginning at location 10) and is stored in memory beginning at location 10. Location 20 contains all zero's.

A flow diagram of this problem is:


START
ASSIGN INDEX VALUE INCREMENT LIMIT
CLEAR ACCUMULATOR TO ZERO
ADD CONTENTS OF EFFECTIVE ADDRESS
TEST INCREMENTED INDEX vs. LIMIT
STORE RESULT
STOP

| | | |
|---|---|---|
| 0010 | AXV 0,2 | Assign index reg. 2 a zero value |
| 0011 | AXI 1,2 | Assign index reg. 2 an increment of one |
| 0012 | AXL 29,2 | Assign index reg. 2 a limit of 29 |
| 0013 | LDA 20 | Clear accumulator to zero's |
| 0014 | ADD 40,2 | Add contents of effective address to accumulator |
| 0015 | JXI 14,2 | Increment index and test |
| 0016 | STA 100 | Store result |
| 0017 | HLT | Stop |

Note that the index register limit is set at one less than the number of data words to be summed. This is done because the new value of the index register must exceed the limit in order to fall out of the ADD loop.

## INDIRECT ADDRESSING

Indirect addressing is a method of relating a particular data address to several commands in the program. It often happens that a piece of data is located in one of several places within the memory. If this piece of data is referenced by several commands, each of these commands will have to be changed as the data appears in different places.

With indirect addressing, only one command need specify the actual address of the data. The other commands can indirectly address the data through the address in this one command.

A command with an indirect address is written just as any normal command. An operator and an address is included. An indirect addressing flag is also written as part of the operator code. This flag indicates that the address specified with this operator is not the address of the data, but is the address of a location in memory which contains the data address.

### Example No. 1

Consider the case where a piece of data is loaded into the accumulator by a command in location 30. The data was found in location 100 and the results are to be placed in 100 after the commands in locations 31 and 32 are obeyed.

```
30    LDA    100

31    ---    ---

32    ---    ---

33    STA*    30
```

The asterisk indicates that the store command in location 33 has an indirect address. The desired storage address is in location 30 and the contents of the accumulator are to be stored in location 100.

If the load command in the example above had been modified by an index register, the store command would have been affected by the same modification.

### Example No. 2

Consider the following with index register 5 having a value of 20.

```
30    LDA    80,5

31    ---    ---

32    ---    ---

33    STA*    30
```

As the load instruction is executed, the value of index register 5 is added to 80 to form an effective address of 100. When the store command is executed, it indirectly uses the effective address in location 30 and stores the contents of the accumulator in 100.

As any command is decoded, its address portion is evaluated first and any index register modification is completed. Next, the indirect addressing indicator is investigated. If there is no indirect addressing, the operator is decoded and executed. If indirect addressing is specified, the address portion and indirect address indicator of the effective address are evaluated as described above. This process continues until a finite effective address is encountered and then the operator is decoded and executed. In this method, indirect addressing takes precedence over index register modification and can be specified to any number of levels.

## OPERATING MODES

DIP can operate in both manual and automatic modes. In the manual mode, commands are given to the interpreter from the typewriter. These commands are executed as they are given, without storage in the computer memory. In this mode, the interpreter makes the computer act much the same as a desk calculator. Each operation is completed before the next is specified. This mode is most useful for small programs.

Some commands, such as jumps, have no meaning in manual mode. These are interpreted as NO OPERATION if they are executed in manual mode.

In automatic mode, the interpreter executes commands stored in memory in a set sequence. The sequence of execution is described below under COMMAND SEQUENCE. Commands may be entered from the typewriter or the paper tape reader for later execution in automatic mode. In this mode, the interpreter acts just as any normal stored program computer. This mode is best used on very intricate or large programs.

Several methods of changing mode are provided. DIP is set to operate in manual mode as it is loaded into the computer. A HALT or BREAKPOINT HALT command executed in automatic mode changes the mode to manual. In addition, a sense switch on the machine console is provided (sense switch 6) to force manual mode at any point when the program is in automatic mode. The occurrence of errors also leaves the interpreter in manual mode. Automatic mode can be established by executing the ENTER AUTOMATIC MODE command.

## COMMAND SEQUENCE

In automatic mode, DIP commands are normally obeyed in the numerical sequence of their memory locations. The interpreter must be told the address of the first command to be executed. After this, the interpreter will automatically obey, in turn, the command in each succeeding address.

In addition, commands are available which cause a jump to a command other than the next one in normal sequence. The interpreter will then obey commands in the normal sequence starting from this new point. These jump commands may be unconditional or may depend on the value of some calculated result.

If desired, the address of a jump command may be remembered by the interpreter so that control can be returned to the original group of commands later in the program. The command giving this capability is called JUMP STORE and is described in Section III. With this command, any number of addresses may be remembered.

The jump commands are useful when the interpreter is executing instructions stored in the memory, that is, when the interpreter is operating in automatic mode. These commands have no meaning and are interpreted as NO OPERATION when executed in manual mode.

The memory of this interpreter is of the "wrap-around" type. Thus, the next address in logical order after the maximum allowable address is 0. In the execution of commands, the same "wrap-around" takes place. A command, other than a jump, in the maximum allowable location will be followed, in normal sequence, by the command in location zero.

## DATA

Data for DIP programs is always decimal numeric and may be entered from the typewriter or paper tape reader in fixed or floating point notation. In the fixed point notation, data takes the following general form:

±NNN.NNN

Up to 12 significant digits may be specified in one data word. In addition, a sign and/or decimal point may be specified. A data word is variable in length and terminated by a tab, carriage return, or stop when read in from the typewriter or paper tape reader.

Computation within the DIP system is always carried out in double precision, floating point, binary notation. For this reason, DIP converts and stores all data entering the machine in this form. However, data communicated between the user and the interpreter and vice-versa is always decimal.

Fixed point data is restricted in that the maximum number expressable in this form is a 12-digit integer. Floating point notation allows the user to represent larger numbers in the machine. Floating point data takes the following general form:

±.NNNNNE±NN

A floating point number has two parts, a fraction and an exponent. The fraction shows the most significant digits of the desired number and appears to the left of the E in the form above. The fraction portion has the sign of the number and a decimal point preceding the most significant digit. The exponent portion is to the right of the E in the form shown and represents the power of 10 to which the fraction is raised.

The exponent also has a sign to indicate positive and negative powers. In DIP, the exponent can range from -75 to +75 and the fraction can be any length up to 12 significant digits. The decimal floating point form shown above is converted to binary for storage and computation within the machine. Data is always read in or out of the machine in decimal form.

The following is a list of rules for data to be used with DIP programs.

1) Input data words must be written in decimal form using fixed or floating point notation.

2) Each data word regardless of notation must be terminated by a tab, carriage return, or stop.

3) In fixed point notation, data without sign or decimal point will be considered to be a positive integer.

4) In floating point notation, either sign unspecified will be assumed positive. If a decimal point is shown, it must precede the most significant digit of the fraction. The E designator must always be present in floating point notation.

5) All data in DIP is represented in double precision, floating point, binary form internally. This implies the limits in accuracy and expressable range included in a floating point binary number with an 8-bit exponent plus sign and a 38-bit fraction plus sign.

## COMMAND STRUCTURE

DIP has a repertoire of 48 commands, falling into seven basic categories. These are: load and store, arithmetic, transcendental, jump, index, control, and input-output. The load and store commands include those which move data to and from the accumulator. The arithmetics perform computations on data in the accumulator. The transcendentals are also arithmetic in nature and include such functions as sine and cosine. The jump commands are concerned with the sequence in which other commands are executed. The control commands serve as aids to the user in controlling the interpreter itself and in program error detection. Input-output commands provide the functions of moving data into and out of the system.

Commands are given to DIP in a format recognized and understood by the interpreter. The command consists of two major elements. These are called the operator and the address. In a few cases, just an operator is sufficient to specify a command; however, the usual form of command has both. Commands are written in the same way whether they are to be typed into the computer, entered on paper tape or executed as they are entered.

The first three characters of any command is the operator. The operator is an abbreviation which serves to tell DIP what operation is to be performed. The operator must always be present in three alphabetic character form.

The next character of the command is considered part of the operator and may or may not be present. This character is an asterisk and indicates indirect addressing. If an asterisk is not present, no indirect addressing takes place.

The operator portion of the command with or without an indirect addressing indicator is terminated by a "tab" or "carriage return" character.

The address portion of a command specifies the location of the data to be operated upon and the index register desired, if any. The address is of variable length and must be written in decimal numeric form following the "tab." Only significant digits of the address need be written. Thus, the address "00039" should be written "39". The maximum allowable address varies with the size of the DDP-24 memory.

If the command is to be index register modified, the address is followed by a comma and the numeric designator (0-9) of the desired index register.

The command terminator is a "carriage return" and it immediately follows the last entry of the command. If the command has no address portion (as in HALT), the operator should be followed by a "carriage return." Spaces within either portion of a command are completely ignored.

Inconsistencies such as the absence of a "tab" between the operator and address, or an invalid operator, etc., will cause the interpreter to cease computation and display an error code indicating what has happened. In the event of an error stop code, the interpreter automatically reverts to manual operation.

A command using all available options would be written in the following form:

ØØØ*tAAAAA,X≠

| | | | |
|---|---|---|---|
| where: | Ø | = | the operation mnemonic |
| | * | = | indirect address indicator |
| | t | = | "tab" for operator delimiter |
| | A | = | variable length decimal address |
| | , | = | index register separator |
| | X | = | the number of the desired index register |
| | ≠ | = | carriage return specifying end of command |

Commands are accepted from the typewriter or the paper tape reader in the above form; they are also displayed on these devices in this form.

A command has one other property which is important to the user. Just as data is located in the machine by an address, so also are commands stored by address. The command address is the location in memory at which the command is stored if the user is operating the interpreter in automatic mode. As DIP commands are read into the machine, the interpreter stores them in a modified form in the machine memory. A single DIP command occupies two 24-bit machine words in the computer. The user need not concern himself with this two computer word storage scheme. Any command or data word is addressable by the decimal address associated with it in the DIP language. The number of addressable locations is a function of the size of the computer memory. Data or commands may be stored at any available address.

# SECTION III

# DIP COMMAND REPERTOIRE

## INTRODUCTION

The following is a list of all DIP commands. Each command is presented in the following form:

LOAD ACCUMULATOR    A or M    LDA

followed by a brief explanation of the operation of the command and any restrictions which apply thereto.

In the example, LOAD ACCUMULATOR is the name of the command. The name of each command specifies its function in brief. The A or M means that the command is valid in either automatic or manual mode. The command format and the method of writing commands are presented in Section II.

LDA is the mnemonic command in the example above. If the address specified in the command is indirect, an asterisk will immediately follow the command mnemonic. If the address is to be index register modified on execution, then the number of the desired index register follows the address separated by a comma. All commands may be modified by indirect addressing and/or an index register.

## LOAD AND STORE

LOAD ACCUMULATOR    A or M    LDA

The contents of the accumulator are replaced by the contents of the effective address. The information in the effective address is not changed.

LOAD ACC. MAGNITUDE    A or M    LDM

The contents of the accumulator are replaced by the positive absolute value of the contents of the effective address. The information in memory remains unchanged.

LOAD ACC. NEGATIVE    A or M    LDN

The contents of the accumulator are replaced by the contents of the effective address with opposite sign. The information in memory is not changed.

STORE ACCUMULATOR A or M STA

The contents of the accumulator are stored at the effective address, replacing the previous contents of the effective address. The contents of the accumulator remain unchanged.

## ARITHMETIC

ADD A or M ADD

The contents of the effective address are added to the contents of the accumulator. The sum replaces the previous contents of the accumulator. The interpreter comes to an error halt if the result exceeds the range of representable numbers $(10^{-75} - 10^{+75})$. See Section IV for a discussion of error halt.

ADD MAGNITUDE A or M ADM

The positive absolute value of the contents of the effective address is added to the contents of the accumulator. The result replaces the previous contents of the accumulator. The interpreter comes to an error halt if the result exceeds the range of representable numbers $(10^{-75} - 10^{+75})$.

SUBTRACT A or M SUB

The contents of the effective address are subtracted from the contents of the accumulator. The result replaces the previous contents of the accumulator. The interpreter comes to an error halt if the result exceeds the range of representable numbers $(10^{-75} - 10^{+75})$.

SUBTRACT MAGNITUDE A or M SBM

The positive absolute value of the contents of the effective address is subtracted from the accumulator. The result replaces the previous contents of the accumulator. The interpreter comes to an error halt if the result exceeds the range of representable numbers $(10^{-75} - 10^{+75})$.

MULTIPLY A or M MPY

The contents of the accumulator are multiplied by the contents of the effective address. The resulting product replaces the previous contents of the accumulator. The interpreter comes to an error halt if the result exceeds the range of representable numbers $(10^{-75} - 10^{+75})$.

DIVIDE A or M DIV

The contents of the accumulator are divided by the contents of the effective address. The quotient replaces the previous contents of the accumulator. The interpreter comes to an error halt if the result exceeds the range of representable numbers $(10^{-75} - 10^{+75})$.

INVERSE DIVIDE A or M IDV

The contents of the effective address are divided by the contents of the accumulator. The quotient replaces the previous contents of the accumulator. The interpreter comes to an error halt if the result exceeds the range of representable numbers $(10^{-75} - 10^{+75})$.

## TRANSCENDENTAL

NATURAL LOGARITHM A or M LGE

This command computes the logarithm to the base e of the contents of the effective address. The result replaces the previous contents of the accumulator. If the contents of the effective address are less than or equal to zero, or if the result exceeds the range of numbers representable in the computer ($10^{-75}$ - $10^{+75}$) the interpreter will come to an error halt. The data in the effective address is left unchanged in either case.

e EXPONENTIAL A or M EXP

e to the power specified by the contents of the effective address is computed. The result replaces the previous contents of the accumulator. Should the result exceed the range of allowable numbers, the interpreter will come to an error halt. The contents of the effective address are not changed.

SQUARE ROOT A or M SRT

This command computes the square root of the contents of the effective address. The result replaces the previous contents of the accumulator. If the contents of the effective address are negative, the interpreter will come to an error halt. The contents of the effective address remain unchanged in any case.

SINE A or M SIN

The effective address is assumed to contain the value of an angle expressed in radians. The sine of the effective address is computed and the result replaces the previous contents of the accumulator. The contents of the effective address are left undisturbed.

COSINE A or M COS

The effective address is assumed to contain the value of an angle expressed in radians. The cosine of the effective address is computed and the result replaces the previous contents of the accumulator. The contents of the effective address are left undisturbed.

ARCTANGENT A or M ATN

This command computes the arctangent of the contents of the effective address. The value of the result is a first or fourth quadrant angle expressed in radians. The result replaces the previous contents of the accumulator. The contents of the effective address are left unchanged.

## JUMP

UNCONDITIONAL JUMP A JMP

This command causes the interpreter to take its next command from the effective address and proceed from there. In manual mode, this command has no meaning and is interpreted as NO OPERATION.

JUMP ON ACC. POSITIVE A JAP

If the sign of the accumulator is positive, the interpreter takes its next command from the effective address and proceeds from there. If the sign is negative, the next sequential command is taken. The sign is left unchanged in either case. In manual mode, this command has no meaning and is interpreted as NO OPERATION.

JUMP ON ACC. POSITIVE OR ZERO A JPZ

If the sign of the accumulator is positive, or if the magnitude of the accumulator is zero, the interpreter takes its next command from the effective address and proceeds from there. If the sign is minus, <u>and</u> the magnitude is not zero, the next sequential command is taken. In either case, the contents of the accumulator are left unchanged. In manual mode, this command has no meaning and is interpreted as NO OPERATION.

JUMP ON ACC. NEGATIVE A JAN

If the sign of the accumulator is negative, the interpreter takes its next command from the effective address and proceeds from there. If the sign is positive, the next command in normal sequence is taken. In either case, the sign is left unchanged. In manual mode, this command has no meaning and is interpreted as NO OPERATION.

JUMP ON ACC. ZERO A JZE

If the magnitude of the contents of the accumulator is zero, the interpreter takes its next command from the effective address and proceeds from there. If the contents are not zero, the interpreter takes the next sequential command. In either case, the contents of the accumulator are left unchanged. In manual mode, this command is meaningless and is interpreted as NO OPERATION.

JUMP STORE A JST

The address of the next command in normal sequence is stored in the location specified by the effective address. The interpreter then takes its next command from the effective address plus one. This command stores the address of the next sequential command for later return. The address is stored in the address portion of the command at the effective address. The remainder of the command at the effective address is left undisturbed. This command has no meaning in manual mode and is interpreted as NO OPERATION.

JUMP RETURN A JRT

The interpreter takes its next command from the location specified in the contents of the effective address. The location specified in the contents of the effective address was placed there by a previously obeyed JUMP STORE command. The contents of the effective address are left unchanged. This command has no meaning in manual mode and is interpreted as NO OPERATION.

## INDEX

ASSIGN INDEX VALUE A or M AXV

The index register specified is set to the value appearing in the address portion of this command. The index register indicator must be present and is used to specify which index register is to be set.

ASSIGN INDEX INCREMENT  A or M  AXI

The increment for the specified index register is set to the value appearing in the address portion of this command. The index register indicator must be present and is used to specify which index register increment is to be set.

ASSIGN INDEX LIMIT  A or M  AXL

The limit for the specified index register is set to the value appearing in the address portion of this command. The index register indicator must be present and is used to specify which index register limit is to be set.

JUMP ON INDEX INCREMENTED  A  JXI

The increment for the specified index register is added to the value of the specified index register. If the new value of the index register is less than or equal to the limit for that index register, the interpreter takes its next command from the effective address. If the new value of the index register is greater than the limit, the next instruction in normal sequence will be taken. The index register indicator must be present and is used to specify which index register is to be incremented and tested. In manual mode, this command has no meaning and is interpreted as NO OPERATION.

## CONTROL

NO OPERATION  A or M  NOP

This command specifies no operation and has the effect of a jump to the next command in normal sequence. This command has no meaning outside of automatic mode. If an address is associated with this command, it is ignored.

BREAKPOINT HALT  A  BHT

If the breakpoint sense switch on the console specified by the address portion of this command is set on, computation is halted. If the sense switch is not set, this command has the effect of a jump to the next command in normal sequence. This command has no meaning outside of the automatic mode; it places the interpreter in the manual mode. Any manual mode command can be executed after this command. To resume automatic mode, an ENTER AUTOMATIC MODE command must be executed. Only sense switches 1 to 4 may be specified in the address portion of this command.

HALT  A  HLT

Computation is halted and the computer is put into manual operating mode. In manual mode, this command has no meaning and will be interpreted as NO OPERATION. An address associated with this command is ignored.

ENTER AUTOMATIC MODE  M  EAM

The computer is placed in automatic mode beginning with the command stored in the effective address location. Computation proceeds automatically until a command is executed which changes the mode back to manual or until sense switch 6 is set.

CLEAR MEMORY      A or M      CLM

This command places zeros into the memory beginning with the effective address location and continuing for the number of successive locations specified in the input limit register. The register must be set prior to this command. If the register is set to zero, no locations will be cleared.

ASSIGN INPUT LIMIT      A or M      AIL

The input limit register is set to the value specified in the address portion of this command. This value is an integer used as a limit on PERMIT TYPE-IN, READ PAPER TAPE, and CLEAR MEMORY commands. Settings of zero and one are not equivalent.

ASSIGN OUTPUT LIMIT      A or M      AOL

The output limit register is set to the value specified in the address portion of this command. This value is an integer used as a limit on paper tape punch commands. Settings of zero and one are not equivalent.

EXECUTE      A      XEC

The command stored in the effective address location is executed. The command following the EXECUTE is normally the next in sequence. If the command executed is a satisfied jump, the interpreter takes the path of the jump. In manual mode, this command has no meaning and is interpreted as NO OPERATION.

## INPUT-OUTPUT

PERMIT TYPE-IN      A or M      PTI

Computation halts and the interpreter sets itself to accept information from the typewriter. The interpreter will accept commands, fixed point data, and floating point data in any order or combination. The number of words to be entered is specified in the input limit register prior to execution of this command. Commands must be terminated by a carriage-return; data words may be terminated by a carriage return, tab, or stop. Information will be stored in successive memory locations beginning at the effective address in this command. The address about to be filled will be typed preceding each type-in. The PERMIT TYPE-IN command will be typed preceding each sequence of type-ins. Spaces within commands or data words will be completely ignored.

The interpreter will automatically return to its previous mode after the last type-in is complete.

CARRIAGE RETURN      A or M      CRT

The paper in the typewriter carriage is automatically positioned by the execution of the number of carriage-returns specified in the address portion of this command.

TAB      A or M      TAB

The paper in the typewriter carriage is automatically positioned by the execution of the number of tabs specified in the address portion of this command.

SPACE A or M SPC

The paper in the typewriter carriage is automatically positioned by the execution of the number of spaces specified in the address portion of this command.

TYPE LAST COMMAND A or M TLC

The last command executed by the interpreter is displayed on the typewriter in normal input format. One type-out only occurs.

READ PAPER TAPE A or M RPT

Punched paper tape containing commands and/or data words is read into the computer memory starting at the effective address. The number of commands or data words to be read-in is determined by the integer in the input limit register. The register must be set prior to execution of this command. Commands must be terminated by a carriage return; data words may be terminated by a carriage return, tab, or stop. Spaces within commands or data words will be completely ignored. Data words may be in fixed or floating point notation.

The interpreter will automatically return to its previous mode after the last word is read.

PUNCH PAPER TAPE A or M PPT

Data and/or commands from memory are punched into paper tape beginning at the effective address. The number of data or command words punched is determined by an integer in the output limit register. The register must be set prior to execution of this command. Data or command words will be terminated by a carriage return. Command words will be punched in normal input format. Data words will be punched in normal input format in floating-point notation.

The interpreter will automatically return to its previous mode after the last word is punched.

TYPE INTEGER NOT FORMATTED A or M TIN

The contents of the effective address are converted to a truncated integer and displayed on the typewriter. Only one integer can be typed with this command (the contents of the output limit register are ignored). Up to 12 digits can be typed as an integer with leading zeros suppressed. The contents of the effective address are unchanged.

TYPE FLOATING FORMATTED A or M TFF

The contents of the effective address are displayed on the typewriter in normal interpreter notation. A data word will be typed in floating-point and a command word will be typed in normal input notation. Each data or command word typed will be preceded by its location and followed by a carriage return. The number of sequential words to be typed starting at the effective address are specified in the output limit register. The contents of the effective address are unchanged.

TYPE FLOATING NOT FORMATTED A or M TFN

The contents of the effective address are displayed on the typewriter in normal interpreter notation. A data word will be typed in floating-point and a command word will be typed in normal input notation. Only one word can be typed with this command (the contents of the output limit register are ignored). The data or command word typed will not be preceded by its location nor followed by a carriage return. The contents of the effective address are unchanged.

TYPE FIXED FORMATTED A or M TXF

The contents of the effective address are converted to a fixed-point number and displayed on the typewriter. Each word typed will be preceded by its location and followed by a carriage return. The number of sequential words to be typed starting at the effective address are specified in the output limit register. The contents of the effective address are unchanged. All fixed-point numbers are 14 characters in length and take the general form: ±IIIII. FFFFFF (where I is the integer portion and F is the fractional portion with leading zeros replaced by spaces). If the word to be typed exceeds this range or it is a command word, the interpreter will give an error indication and switch to the manual mode.

TYPE FIXED NOT FORMATTED A or M TXN

The contents of the effective address are converted to a fixed-point number and displayed on the typewriter. Only one word can be typed with this command (the contents of the output limit register are ignored). The word typed will not be preceded by its location nor followed by a carriage return. The contents of the effective address are unchanged. All fixed-point numbers are 14 characters in length and take the general form: ±IIIII. FFFFFF (where I is the integer portion and F is the fractional portion) with leading zeros replaced by spaces. If the word to be typed exceeds this range or it is a command word, the interpreter will give an error indication and switch to the manual mode.

# SECTION IV

# DIP OPERATING PROCEDURES

## MANUAL MODE

In manual mode, commands are executed as they are typed into the interpreter. The function of each command is completed before the next command is accepted from the typewriter.

Each command should be written in the format in which it will be typed including the TAB operation delimiter and the CARRIAGE RETURN command terminator. The data for the program is entered into the machine through the typewriter or the paper tape reader. Spaces and delete codes within commands or data words are completely ignored by DIP. A command or data type-in can be terminated at any time without storage or execution by typing a dollar sign ($).

After the program and the data are written down and the user has firmly in mind each step in his program, he is ready to effectively use the computer. When the interpreter is loaded into the machine, it is set to operate in the manual mode. It types an indication that it is ready and stops awaiting the user's first command. The user will enter his data by first executing a command setting the limit register and then executing the desired input command. The action of these commands is described in the command repertoire. After the data is entered, the interpreter will accept commands and perform operations on the data as it is instructed by the user. As operation is completed, the interpreter will type an indication that it is ready for its next command. In manual mode, completion of the program need not be characterized by a HALT. The interpreter is in a HALT state at the end of each command.

In the event an invalid command is typed in or some other inconsistency occurs, the interpreter will type out the command in question followed by an error code indicating what is wrong. After the error type-out, the interpreter will type a ready indicator and stop to await the next action of the user. The command in error is not executed, and the contents of all registers remain unchanged.

## AUTOMATIC MODE

In automatic mode, the interpreter obeys commands stored in the memory in a set sequence in turn. In this mode, the interpreter acts like a stored program computer and performs each operation automatically.

Programs to be executed in automatic mode are usually longer and more involved than those executed in manual mode. Some additional power of the interpreter is also available to the user in automatic mode. For these reasons, it is very important that the program be thoroughly checked by the programmer before it is given to the machine.

The data and the program are entered into the machine through the paper tape reader or the typewriter. There is no difference in commands written for manual or automatic mode. The user should remember that in automatic mode, commands have addresses just as data words and the program must be loaded into the proper locations in order to operate properly.

As in manual mode, the interpreter is loaded, sets itself to operate in manual mode, types a ready indicator, and stops awaiting the user's first command. The initial commands executed will be in manual mode and include those necessary to load the program and set any conditions that the program assumes exist at the time it is started. Data may or may not be loaded depending on whether the program includes this function. After the preliminaries have been completed, the user will execute an ENTER AUTOMATIC MODE command. The interpreter will change modes and execute the program beginning at the command in the location specified. The interpreter will continue to operate in automatic mode until a HALT or BREAKPOINT HALT or an error is encountered. Any of these conditions will cause a halt in computation and a change to manual mode. In addition, manual mode can be forced at the end of any command by setting a sense switch on the computer console.

Sense switches 5 and 6 play an important role in the operation of the interpreter in automatic mode. Switch numbers 1 to 4 are the breakpoint enable switches and are used in conjunction with the BREAKPOINT HALT command. Sense switch number 5 controls the operation of the trace function in automatic mode and sense switch number 6 is an override switch used to force manual mode at any time in a program operating in automatic mode.

An attempt to execute an invalid or improper command in automatic mode will result in the same interpreter action as in manual mode. The methods of locating programming errors are somewhat different in automatic mode. The trace function is very helpful in this mode. The trace is discussed below.

## PROGRAM CHECKOUT

DIP language contains commands specifically designed to aid the user in interrogating the interpreter. These are useful in detecting and correcting errors made in writing the program.

Two kinds of programming errors are possible when using DIP. These are: those which affect the operation of DIP itself and those in which DIP operates properly and produces erroneous results. The interpreter will recognize and halt execution of any command which threatens to disable it. An attempt at any of the following will result in such a halt:

1) Any internal computation whose result exceeds the range $10^{-75}$ - $10^{+75}$.

2) Square root of a negative number.

3) Division by zero.

4) Log of zero or log of a negative number.

5) Execution of any command invalid in operator, address, or modification.

Upon encountering any of the above conditions, DIP will type the command (and its location), causing the error and an error code indicating the type of error. The interpreter will halt and set itself to manual mode to allow the user to investigate the conditions causing this error. A complete list of error codes and their meanings is shown in Appendix C.

Completed programs whose results are erroneous come from poor logic or improperly coded commands. This type of error is not identifiable by DIP and it is up to the user to obtain enough information from the computer upon completion of the program to insure an accurate answer or to locate the commands which caused the error.

The following is a discussion of some of the methods used in program checkout. The methods outlined are mainly intended for checkout of programs designed to operate in automatic mode. The most appropriate techniques of program checkout in any given case are dependent on many factors. As the user gains experience in programming, he will develop methods which will supplement his thought processes in debugging problems.

## MEMORY INVESTIGATION

Much of error detection is best done in manual operating mode. The TYPE commands allow the user to display the key commands and data within the stored program.

For errors which appear deep in the program, the BREAKPOINT HALT command can be inserted in the program so that it can be stopped to permit investigation before the error actually occurs.

The contents of the accumulator can be displayed by manually executing a set of commands which will store the contents in memory and type from that location.

## TRACE

The trace is a tool to aid checkout of programs operating in automatic mode. The trace displays the results of each command executed on the typewriter. In this way, the user can obtain a picture of all events occurring in the interpreter while the program is in actual operation.

To use the trace, the program is run, using its normal data, with sense switch number 5 set "on." As each command of the program is obeyed by the interpreter, the results are displayed on the typewriter. The results include: the command with modification completed, its location, the contents of the effective address, and the contents of the accumulator after execution. On index register and limit register commands, the contents of the registers will be shown.

The trace may be engaged at any point or number of points within the program. The sense switch turns this function on and off. Particular areas to be traced can be located with breakpoints.

The trace is most helpful in locating such programming errors as: unending loops, transposed digits in command addresses, and errors caused by peculiarities of the data.

## BREAKPOINT

The breakpoint halt command allows the user to segment his program for checkout. This command has the capacity to take on two functions. It can be a HALT and "enter manual mode" or a "do nothing" command, depending upon the setting of console sense switch numbers 1 to 4.

If the specified sense switch is on, the interpreter will halt on the breakpoint command and set itself to manual mode. If the switch is off, the command will be executed as a NO OPERATION and no halt will occur.

Breakpoint halts coded into long programs allow the user to halt at the end of each logical phase of the program and check for errors in that phase. When all phases are checked out, the user can release the breakpoint (sense switch) causing the halt to become a NO OPERATION.

## APPENDIX A

## FUNCTIONAL COMMAND LIST

| Mnemonic | Name | Mode | Page |
|---|---|---|---|
| LOAD AND STORE | | | |
| LDA | Load Accumulator | A or M | 3-1 |
| LDM | Load Acc. Magnitude | A or M | 3-1 |
| LDN | Load Acc. Negative | A or M | 3-1 |
| STA | Store Accumulator | A or M | 3-2 |
| ARITHMETIC | | | |
| ADD | Add | A or M | 3-2 |
| ADM | Add Magnitude | A or M | 3-2 |
| SUB | Subtract | A or M | 3-2 |
| SBM | Subtract Magnitude | A or M | 3-2 |
| MPY | Multiply | A or M | 3-2 |
| DIV | Divide | A or M | 3-2 |
| IDV | Inverse Divide | A or M | 3-2 |
| TRANSCENDENTAL | | | |
| LGE | Natural Logarithm | A or M | 3-3 |
| EXP | e Exponential | A or M | 3-3 |
| SRT | Square Root | A or M | 3-3 |
| SIN | Sine | A or M | 3-3 |
| COS | Cosine | A or M | 3-3 |
| ATN | Arctangent | A or M | 3-3 |
| JUMP | | | |
| JMP | Unconditional | A | 3-3 |
| JAP | Jump on Acc. Positive | A | 3-4 |

| Mnemonic | Name | Mode | Page |
|---|---|---|---|
| JUMP (continued) | | | |
| JPZ | Jump on Acc. Positive or Zero | A | 3-4 |
| JAN | Jump on Acc. Negative | A | 3-4 |
| JZE | Jump on Acc. Zero | A | 3-4 |
| JST | Jump Store | A | 3-4 |
| JRT | Jump Return | A | 3-4 |
| INDEX | | | |
| AXV | Assign Index Value | A or M | 3-4 |
| AXI | Assign Index Increment | A or M | 3-5 |
| AXL | Assign Index Limit | A or M | 3-5 |
| JXI | Jump on Index Incremented | A | 3-5 |
| CONTROL | | | |
| NOP | No Operation | A or M | 3-5 |
| BHT | Breakpoint Halt | A | 3-5 |
| HLT | Halt | A | 3-5 |
| EAM | Enter Automatic Mode | M | 3-5 |
| CLM | Clear Memory | A or M | 3-6 |
| AIL | Assign Input Limit | A or M | 3-6 |
| AOL | Assign Output Limit | A or M | 3-6 |
| XEC | Execute | A | 3-6 |
| INPUT/OUTPUT | | | |
| PTI | Permit Type-In | A or M | 3-6 |
| CRT | Carriage Return | A or M | 3-6 |
| TAB | Tab | A or M | 3-6 |
| SPC | Space | A or M | 3-7 |
| TLC | Type Last Command | A or M | 3-7 |
| RPT | Read Paper Tape | A or M | 3-7 |
| PPT | Punch Paper Tape | A or M | 3-7 |
| TIN | Type Integer Not Formatted | A or M | 3-7 |

| Mnemonic | Name | Mode | Page |
|---|---|---|---|
| INPUT/OUTPUT (continued) | | | |
| TFF | Type Floating Formatted | A or M | 3-7 |
| TFN | Type Floating Not Formatted | A or M | 3-7 |
| TXF | Type Fixed Formatted | A or M | 3-8 |
| TXN | Type Fixed Not Formatted | A or M | 3-8 |

## APPENDIX B

## ALPHABETIC COMMAND LIST

| Mnemonic | Name | Mode | Page |
|---|---|---|---|
| ADD | Add | A or M | 3-2 |
| ADM | Add Magnitude | A or M | 3-2 |
| AIL | Assign Input Limit | A or M | 3-6 |
| AOL | Assign Output Limit | A or M | 3-6 |
| ATN | Arctangent | A or M | 3-3 |
| AXI | Assign Index Increment | A or M | 3-5 |
| AXL | Assign Index Limit | A or M | 3-5 |
| AXV | Assign Index Value | A or M | 3-4 |
| BHT | Breakpoint Halt | A | 3-5 |
| CLM | Clear Memory | A or M | 3-6 |
| COS | Cosine | A or M | 3-3 |
| CRT | Carriage Return | A or M | 3-6 |
| DIV | Divide | A or M | 3-2 |
| EAM | Enter Automatic Mode | M | 3-5 |
| EXP | e Exponential | A or M | 3-3 |
| HLT | Halt | A | 3-5 |
| IDV | Inverse Divide | A or M | 3-2 |
| JAN | Jump on Acc. Negative | A | 3-4 |
| JAP | Jump on Acc. Positive | A | 3-4 |
| JMP | Unconditional Jump | A | 3-3 |
| JPZ | Jump on Acc. Positive or Zero | A | 3-4 |
| JRT | Jump Return | A | 3-4 |
| JST | Jump Store | A | 3-4 |

| Mnemonic | Name | Mode | Page |
|---|---|---|---|
| JXI | Jump on Index Incremented | A | 3-5 |
| JZE | Jump on Acc. Zero | A | 3-4 |
| LDA | Load Accumulator | A or M | 3-1 |
| LDM | Load Acc. Magnitude | A or M | 3-1 |
| LDN | Load Acc. Negative | A or M | 3-1 |
| LGE | Nutural Logarithm | A or M | 3-3 |
| MPY | Multiply | A or M | 3-2 |
| NOP | No Operation | A or M | 3-5 |
| PPT | Punch Paper Tape | A or M | 3-7 |
| PTI | Permit Type-In | A or M | 3-6 |
| RPT | Read Paper Tape | A or M | 3-7 |
| SBM | Subtract Magnitude | A or M | 3-2 |
| SIN | Sine | A or M | 3-3 |
| SPC | Space | A or M | 3-7 |
| SRT | Square Root | A or M | 3-3 |
| STA | Store Accumulator | A or M | 3-2 |
| SUB | Subtract | A or M | 3-2 |
| TAB | Tab | A or M | 3-6 |
| TIN | Type Integer Not Formatted | A or M | 3-7 |
| TFF | Type Floating Formatted | A or M | 3-7 |
| TFN | Type Floating Not Formatted | A or M | 3-7 |
| TLC | Type Last Command | A or M | 3-7 |
| TXF | Type Fixed Formatted | A or M | 3-8 |
| TXN | Type Fixed Not Formatted | A or M | 3-8 |
| XEC | Execute | A | 3-6 |

# APPENDIX C

## PROGRAMMING ERROR CODES

| Code | Error |
|---|---|
| 0 | Invalid operator |
| A | Invalid address |
| I | Invalid indirect address |
| X | Invalid index |
| V | Result exceeds the range of allowable numbers |
| S | Square root of a negative number |
| D | Division by zero |
| L | Logarithm of zero or a negative number |

# APPENDIX D

## TYPEWRITER CODES

| OCTAL CODE | TYPEWRITER L/C | U/C | PAPER TAPE | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 8 | 7 | 6 | P | 4 | S | 3 | 2 | 1 |
| 00 | Ø | b | | | | O | | • | | | |
| 01 | 1 | | | | | | | • | | | O |
| 02 | 2 | | | | | | | • | | O | |
| 03 | 3 | | | | | O | | • | | O | O |
| 04 | 4 | : | | | | | | • | O | | |
| 05 | 5 | @ | | | | O | | • | O | | O |
| 06 | 6 | √ | | | | O | | • | O | O | |
| 07 | 7 | > | | | | | | • | O | O | O |
| 10 | 8 | | | | | | O | • | | | |
| 11 | 9 | | | | | O | O | • | | | O |
| 13 | # | - | | | | | O | • | | O | O |
| 20 | * | ¢ | | | O | | | • | | | |
| 21 | / | | | | O | O | | • | | | O |
| 22 | S | | | | O | O | | • | | O | |
| 23 | T | | | | O | | | • | | O | O |
| 24 | U | = | | | O | O | | • | O | | |
| 25 | V | % | | | O | | | • | O | | O |
| 26 | W | " | | | O | | | • | O | O | |
| 27 | X | ' | | | O | O | | • | O | O | O |
| 30 | Y | | | | O | O | O | • | | | |
| 31 | Z | | | | O | | O | • | | | O |
| 33 | ' | | | | O | O | O | • | | O | O |
| 40 | - | - | | O | | | | • | | | |
| 41 | J | | | O | | O | | • | | | O |
| 42 | K | | | O | | O | | • | | O | |
| 43 | L | | | O | | | | • | | O | O |
| 44 | M | ) | | O | | O | | • | O | | |
| 45 | N | * | | O | | | | • | O | | O |
| 46 | O | △ | | O | | | | • | O | O | |
| 47 | P | ; | | O | | O | | • | O | O | O |
| 50 | Q | | | O | | O | O | • | | | |
| 51 | R | | | O | | | O | • | | | O |
| 52 | tab | | | O | | | O | • | | O | |
| 53 | $ | | | O | | O | O | • | | O | O |
| 54 | backspace | ⩗ | | O | | | O | • | O | | |
| 56 | space | | | O | | O | O | • | O | O | |
| 60 | & | & | | O | O | O | | • | | | |
| 61 | A | | | O | O | | | • | | | O |
| 62 | B | | | O | O | | | • | | O | |
| 63 | C | | | O | O | O | | • | | O | O |
| 64 | D | ( | | O | O | | | • | O | | |
| 65 | E | □ | | O | O | O | | • | O | | O |
| 66 | F | / | | O | O | O | | • | O | O | |
| 67 | G | < | | O | O | | | • | O | O | O |
| 70 | H | | | O | O | | O | • | | | |
| 71 | I | | | O | O | O | O | • | | | O |
| 73 | . | V | | O | O | | O | • | | O | O |
| 74 | lower shift | | | O | O | O | O | • | O | | |
| 75 | upper shift | | | O | O | | O | • | O | | O |
| 76 | car. return | | | O | O | | O | • | O | O | |
| 77 | line feed | | | O | O | O | O | • | O | O | O |
| stop | backspace | | O | O | | O | O | • | O | | |