503
COMPUTER
MANUAL

VOL. 2

B

503
COMPUTER
MANUAL

INTRODUCTION

This volume is intended to provide the programmer with all the information necessary to write programs for the 503. For a full account of the electronics, registers and similar facilities, the interested reader is referred to Volume 4 of the Manual. Certain operating instructions are included in this Volume, but Volume 3 of the Manual is intended for the operator of the computer.

The programmer or systems analyst using the 503 has available to him, in addition to the hardware of the machine itself, a considerable range of carefully planned programming systems, including internationally accepted problem–oriented programming languages, and at a more machine–oriented level, such items as macro–instructions for using peripheral equipment with maximum efficiency.

Such programming systems are made possible largely by the greatly increased basic speed of the machine relative to earlier electronic computers. They represent great advances in the ease of programming and the range of problems susceptible to data processing.

It is for the user to decide of which, if any, of these systems he wishes to use. The basic instructions of the machine are also described in full detail.

503 TECHNICAL MANUAL

VOLUME 2: PROGRAMMING INFORMATION

CONTENTS LIST

(Issue 4)

503 TECHNICAL MANUAL

VOLUME 2: PROGRAMMING INFORMATION

CONTENTS LIST

(*Issue* 2)

503 COMPUTER
RESERVED AREA PROGRAM

April 1964

Issue 2

ELLIOTT
SCIENTIFIC COMPUTING DIVISION
Elliott Brothers (London) Limited
Elstree Way, Borehamwood, Hertfordshire, England

CONTENTS LIST

1. INTRODUCTION—FUNCTION OF THE PROGRAM

The Reserved Area Program (RAP) provides the principle method of Manual Control for the 503 Computer. It is read into locations 7936-8165 of the computer store by the fixed instructions contained in locations 0-3 and uses locations 7886-7935 for workspace. The locations 7936-8191 form the Reserved Area of the store and the protection of this area is controlled through the use of the NO PROTN button on the control console (see 1.2.2).

RAP interprets and executes control messages typed in by means of a directly coupled electric typewriter. The DEMAND light on the control console indicates when a message is expected. It is lit either when the operator depresses the MESSAGE button or due to a demand by a main program.

The position of programs in the store is dictated by pointers held by RAP which indicate the first free (FF) and last free (LF) locations of the available store. RAP contains an Input routine which reads a relocatable program tape and places the program in the first available space at either end of the available store. Provision is made for several programs to be in the computer store at the same time, and communication is facilitated by means of a system to which each program is automatically linked on being stored.

Each program must be given a name, which is used for all communication with the program. A program head of five locations is assigned to each program which provides RAP with essential information about each program, including a means of determining the position of each of the programs in the store. The need for the user to know the stored position of a program is thus obviated.

A facility is also provided, through the use of pointers stored in the program head, which enables a sum check to be made on that section of a program expected to remain unaltered. RAP, on attempting entry to a program, first performs this sum check where possible and then enters the program unless the sum check fails. When the running of a program is completed, control of the computer returns to RAP.

RAP is the most fundamental program on the 503, as the programming systems ALGOL, SAP and AUTOCODE all function under its control.

2. CONTROL MESSAGES

The following messages can be typed on the control typewriter:

2.1a

IN.

This instruction typed on the typewriter when the DEMAND lamp is lit causes a specially prepared relocatable binary tape (see below) to be input from whichever tape reader is selected as reader 1.

Unless a message parameter gives indication to the contrary, the program being input is automatically placed in the first available space. Whether it is placed at the upper or lower end of the free store or in a fixed position is a decision which is taken when the binary tape is produced (see Section 8 for further details). When a program has been successfully input by RAP, the program name is output on the same line as the input message.

On entry to the input routine, the eight bits of the Interrupt Mask Register are cleared and the Interrupt Permit Register is set to one (see 1.2.3), so that Manual Interrupts are allowed but Normal Interrupts (from any other program) are inhibited.

Note that each control message must be followed by a full stop.

2.1b

IN; *N*. (where *N* is a number)

This instruction reads a special binary tape whose position in the store is optional. The integer *N* specifies where the program is to be placed.

2.2a

CONT.

This instruction continues a program from the point at which it was left to obey a Manual Interrupt.

2.2b

CONT; ERRINT.

Continues a program after an Error Interrupt.

The last program to be interrupted will be continued from the point of interruption, even if a tape was being input when the Interrupt occurred.

2.3

RESET.

This instruction has the effect of clearing the main store with the exception of the RAP workspace. It also sets the FF (first-free) and LF (last-free) pointers of the store, which indicate where the available store begins and ends (see under 7 below), to their minimum and maximum. The RAP pointer (see below) is set to point to RAP itself.

2.4

LIST.

This prints out on the output writer a list of all programs in the store in chronological order beginning with the last to be stored. After an extra line feed, the size of the available store is printed as a four-digit integer with no suppression of leading zeros. If a program is found to be stored incorrectly as a result of sum check failure, then the program name is followed by an asterisk (see 3.1.4.1 for possible action to be taken).

2.5

CANCEL.

This effectively removes the last program placed in the store by backdating the RAP pointer and the FF and/or LF pointers.

2.6

CANCEL; *NAME.*

In this message *NAME* specifies and is replaced by the name of the program to be removed, e.g. CANCEL; SAP. Not only this program is removed, but all the programs input after it (i.e. named before it in the List). Before the beginning of the backdating process, a check is made to ensure that the named program is present (to guard against possible spelling mistakes). NOPROG is displayed if the program is absent.

2.7

FREE STORE.

This prints the size of the available store on the output writer as a four-digit integer with no suppression of leading zeros.

2.8a

NAME.

This transfers control to the program called *NAME.* The computer only notes the first six non-space characters of each message which may be terminated by a semi-colon or a full stop.

2.8b

N. **and** *N;* **S.** (where *N* is a number)

N. causes control to be transferred to the first instruction in location *N* and *N;* S. causes control to be transferred to the second instruction in location *N.* These messages are only used in special cases, usually in conjunction with an IN; *N.* message.

3. TYPING INSTRUCTIONS

(a) When a message is being typed, any of the upper case characters may be used with the exception of =, but only the letters may be used from the lower case. These restrictions enable messages to be typed more easily.

(b) The symbols + and − are only used to precede integers (+ is optional).

(c) The first character of an identifier must be a letter (upper or lower case). The rest of the identifier may be numerals and/or upper or lower case letters.
(N.B. 'ALGOL' is treated as distinct from 'Algol').

(d) The semi-colon is used as an end of word symbol.

(e) The full stop denotes the end of the message (and also the end of the last word of the message).

(f) The vertical bar is a special non-acceptable character. This may be typed to cancel a partially typed message, which must then be restarted from the beginning. This is the recognised way of cancelling a partially typed message. Once the terminating full stop or semi-colon has been typed the message is acted upon and cannot be cancelled.

(g) All other characters, apart from 'space' which is ignored, are treated as non-acceptable, and typing them is treated as a message error. RAP outputs a vertical bar with the capital letter X superimposed to form a large asterisk. If a message to the computer is intended, it must be restarted from the beginning, but this is not a recommended way of cancelling a message.

(h) In addition, the input writer may be used as a straightforward typewriter in order that comments may be made by the operator. The comment may be composed of any of the admissible alphanumeric characters and the 'space' character and should be terminated with a vertical bar, i.e. comments are printed out by the typewriter but not accepted by the computer since they are treated as partially typed messages cancelled by the non-acceptable character, vertical bar.

All input messages appear in red on the paper in the typewriter.

4. DISPLAYED MESSAGES

All messages from the computer are displayed on the output writer in black.

The standard messages are:

(1) **Large Asterisk** $*$ Explained under 3(g) above. This signifies that a non-acceptable character has been input and that the message has accordingly been cancelled.

(2) **NOPROG** This is output after a search routine has been initiated by an input instruction transferring control to a program, or by a CANCEL; *NAME.* instruction, or otherwise. It signifies that the named program is not available.

(3) **UNCHEK** This indicates that the program is not sumcheckable. This is not strictly an error but serves as an indication that a program will not be protected on entry. It is output on the same line as the program name only on input, i.e. after an IN. instruction has been read.

(4) *NAME* The program name is normally displayed after input of a binary tape has taken place correctly.

(5) **ERRSUM** This is displayed to indicate that a program to which it is desired to transfer control is stored with an incorrect sumcheck, i.e. has been corrupted. Corrective action should be taken by the operator. If it occurs during input of a program tape, the tape should be read in again. If it is displayed on attempting entry to a program a CANCEL; *NAME.* instruction should be typed. The program may then be read in again. If this is not convenient, a duplicate copy of the program may be read in. (See 9(i)).

(6) $>$ This symbol is output to indicate that entry into a program named by an identifier has taken place.

(7) ? This symbol is a response to a MANUAL INTERRUPT by depression of the MESSAGE button, and indicates that the Manual Interrupt is accepted and RAP awaits a message. The symbol is always output on a new line.

(8) **NOROOM** This indicates that there is not enough space in the available store for the program being input.

The operator should take corrective action, e.g. cancel a program or programs until the available store is large enough.

(9) **Small Asterisk** * This is the asterisk as provided by the typewriter, not the symbol capital X combined with a vertical bar (see 1) above.

It appears after the program name as a response to LIST where a program is in store but found to be corrupt. Various types of corrective action can be taken by the operator.

(i) If store space is of no account, read the program in again to the next available space. The corrupt program is never picked up by RAP since the last program placed is always the first examined.

(ii) If the programs input after the corrupt program (i.e. above it in the list) are no longer required, type CANCEL; *NAME.*. The program may be read in again subsequently.

(10) **END** This indicates that STOP has been reached in the program and control transferred to RAP to await a message.

(11) **ERRINT** This signifies an Error Interrupt and is followed by a single integer (normally 1, 3, 4 or 5) or possibly a combination of such integers. If followed by

1 This indicates floating point overflow. The instruction causing overflow is completed before error interrupt takes place.

3 Parity error in the main store. The instruction or autonomous transfer during which the error occurs is completed before error interrupt takes place.

4 Attempted use of an unavailable peripheral device.

5 Attempted impermissible reference to the Reserved Area. The instruction containing the impermissible reference is not obeyed.

These integers are obtained by reading the appropriate location on interrupt (L + 3, see 1.2.5).

After RAP has output ERRINT n it searches for a program ERRINT and if no such program exists, outputs NOPROG. If there is a program ERRINT (usually a short program to facilitate further diagnosis of the error), control is transferred to it.

Note that ERRINT 2 occurs on a control shut-down due to a power failure. The appropriate bit in location L + 3 is set to 1 but there is not sufficient time to print out a message to this effect.

5. CONTROL OF INTERRUPTS

Each interrupt channel has a location associated with it in the Reserved Area. However, in order that the destination of interrupts from Channels 2, 3 and 4 (Tape Readers, Punches and digital plotter, and Peripherals) may be controlled by a main store program when the Reserved Area is protected, a SWITCH TABLE is provided outside the Reserved Area. The switch table occupies fifteen locations, five for each of the three interrupts mentioned above. Four locations are reserved for the storage of the registers on interrupt and one contains the transfer instruction for the interrupt routine, which may be set by the main program. The location of the transfer instruction for each of the interrupt channels is as follows:

Tape readers	INT TR
Punches and plotter	INT PUN
Peripherals	INT PER

(The absolute locations are given in the Appendix)

Thus the contents of the five locations in the switch table for tape reader interrupt are:

INT TR) transfer instruction
C (modifier)
C (Accumulator)
C (Aux Register)
C (SCR and OFR)

and similarly for punches and plotter, and peripherals.

The interrupt locations in the Reserved Area hold the following pointers to the switch table:

8172) 00 INT TR : 00 INT TR + 1
8171) 00 INT PUN : 00 INT PUN + 1
8170) 00 INT PER : 00 INT PER + 1

When a tape reader interrupt occurs, the contents of the registers are stored in INT TR + 1 to INT TR + 4 and control is then transferred to INT TR, which holds the transfer instruction to the interrupt routine. At the end of the routine the 66 instruction restores the registers and returns control to the main program.

When RAP is first read into the store there are standard transfer instructions in INT TR, INT PUN and INT PER which ensure that although the registers are stored, the interrupt does not have any effect since control is immediately transferred back to the main program. If interrupts are required, the contents of the appropriate location must be set by program.

e.g. To ensure that a tape reader interrupt causes entry to a main program block called TAPE, the following instructions are required:

```
30 < 40 TAPE : >
20 INT TR
```

The standard settings of the switch table transfer locations are only preserved if the user resets the appropriate location. The instructions to restore the standard setting in the tape reader interrupt location are:

```
30 < 00 8172/66 0 >
20 INT TR
```

5.1 Use of Interrupt Mask

A location in the RAP workspace known as IMASK is used to preserve a record of how the Interrupt Mask Register was set (see 1.2.3 and 1.2.4 of the Manual).

Whenever RAP is read into the store or whenever a program is entered, the contents of IMASK and of the Mask Register are set to a standard value which in the case of RAP (Issue 2) is zero; (if a different standard setting for IMASK is required, the 503 Librarian can give further information).

Whenever a RAP subroutine is called by a main store program a copy of the Interrupt Mask Register should be in IMASK. As it is essential that interrupts be prevented during such a subroutine, the overall inhibition is set by RAP on entry. Exit from the RAP subroutine resets the Mask Register in accordance with the contents of IMASK and also sets the Permit bit to 1.

If the programmer wishes to change the setting of the Mask Register for an interrupt under his control, then he must ensure that the setting of the other bits in the Mask Register is preserved and that IMASK contains a copy of this setting. The procedure to be adopted is best illustrated by examples:

A. To allow Interrupt on the tape reader (Channel 2):

	IMASK setting	*Mask reg. setting*
72 0	. . . 000XXXX XXXX	0000 0000
30 < + 191 >		
23 IMASK	. . . 000X0XX XXXX	0000 0000
30 < + 64 >		
24 IMASK/	. . . 000X1XX XXXX	0000 0000
72 256		X1XX XXXX

where X = the original bit state

B. To prevent interrupts on Peripheral devices (Channel 4):

	IMASK setting	*Mask reg. setting*
72 0	. . . 00XXXX XXXX	0000 0000
30 < + 239 >		
23 IMASK/	. . . 00XXX0 XXXX	0000 0000
72 0		XXX0 XXXX

N.B. Reference to IMASK must be made by using the absolute value 7886 until SAP Issue 3 makes provision for mnemonic reference.

When a routine has been entered via an interrupt, the programmer normally wishes for any further interrupts to be inhibited until the routine has finished (i.e. a return to the main program before any other interrupts are permitted). Since entry to the routine automatically clears the Permit bit, no further action is necessary to set inhibition. To ensure that the main program is re-entered before any other interrupts occur, the inhibition should be released by using the 66 order and not the 72 256 order, e.g.

```
67 IMASK
72 0
66 return
```

If the 66 order were not used, inhibition would be released before the registers were restored; thus another interrupt could occur before the return to the main program and overwrite the information stored in the switch table, so destroying the return link.

If, however, another selected interrupt is required during the interrupt routine, the inhibition should be released thus:

```
67 IMASK
72 256
```

6. EXAMPLE

This is a conceivable output on the typewriter. Underlining indicates what appears on the record in red.

```
?
C H Hagerty 8 10 a m 22 1 64 *
LIST.
CHECK
SAP*
PRINT

5649
CANCEL; SAP.
Comment Check and SAP have been removed *
IN. ALGOL UNCHEK
IN. DRS
Comment The program OBJECT for translation by ALGOL is now under the reader UNCHEK
        means that no sum check is possible *
```

ALGOL. > OBJECT FREE STORE 4495-6600
END
LIST.
OBJECT
DRS
ALGOL
PRINT

2105
OBJECT. >
?
Comment Message button pressed to place the correct data tape under the reader *
CONT.
END OF THE PROGRAM

7. SUBROUTINES AVAILABLE TO THE PROGRAMMER

The following RAP subroutines may be used by a program in the main store. Each has a special external entry for this purpose in order that they may be protected in case of Manual Interrupt.

They are:

(1) **Read word from typewriter**

The entry instruction is **RAPread.**

When exit has been made from this subroutine the word which has just been read is held in location RAPword. If the overflow register is set on exit, an alphanumeric word has been read. The character on which exit is made is held in location RAPsep and the accumulator. If the overflow register is not set, then an integer has been read. The terminator is held in the accumulator only. When an unacceptable character is typed, a solid triangle ▲ is displayed on the output writer. The complete word must then be repeated.

(2) **Print alphanumeric word on output writer**

The entry instructions are **RAP1print** and **RAPprint.**

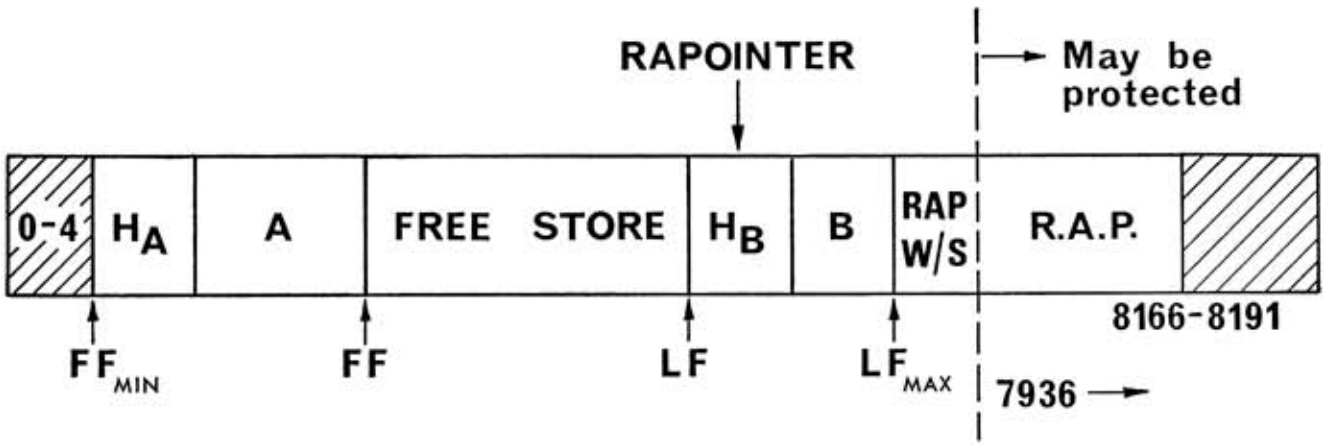
The former is for printing the word on a new line and the latter for printing it on the same line (see the section on relocatable binary for the form of the word). For both, the word is held packed in the accumulator. The packed form is given in Section 8.

(3) **Search for named program**

The entry instruction is **RAPsearch.**

This subroutine is entered with the name of the required program held in the accumulator as shown in the diagram on Page 9. On exit from the subroutine the address of the first location of the program head is in the accumulator unless the program is not available, in which case the content of the accumulator is zero.

8. METHOD OF STORAGE ALLOCATION



In the above diagram, assume that A and B are the first and second programs to be placed by RAP and that HA and HB are the locations which contain their respective 'Heads'. The five 'Head' locations contain the following details:

0, Internal checksum of program.

1, Main entry to program. Pointer to previous program.

2, Name of program.

3, Checkable address points (zero if no check possible).

4, Previous first-free (FF) and last free (LF) location points.

Shortage of space made it necessary that the RAP workspace should be placed immediately outside the Reserved Area where it is 'semi-protected' by the RAP system, i.e. the maximum LF pointer points to one location before the RAP workspace, so it will not be overwritten by a program being read into the store. There is, however, no protection against the program itself causing a location of the RAP workspace to be overwritten.

RAP has available the first free location, last free location and a pointer to the last program placed. To find a named program, the search routine picks up the RAP pointer and steps through the programs in the store until either the required program is found or RAP is reached again, in which case NOPROG is displayed on the output writer and RAP then awaits the next typewriter message.

9. RELOCATABLE BINARY

This is a special form of binary tape which is designed to be read in under the Standard Input routine. Programs prepared in this form may be placed in the free store either from the first free location upwards (i.e. FF+) or up to the last free location (i.e. LF—). A translator exists which will convert programs from their TI form to this binary form. Sum-checked binary versions of existing 803 Library programs may also be converted with the Translator. These, however, are fixed programs and are dealt with more fully later in this Section.

The decision as to whether a program is placed FF+ or LF— is taken when the program is coded since this is determined by the presence of 'action-words' (i.e. instruction-pairs which are immediately obeyed) which precede and succeed the program proper.

The Standard Input routine simply reads a word, modifies it as necessary, and detects whether it is to be placed in the store or obeyed immediately. The action-words are used to perform the house-

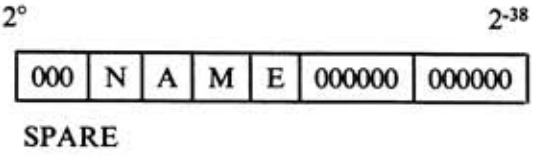
keeping functions appropriate to the particular type of program being input. These functions are described in detail later in this Section.

The first non-blank character of all tapes of this binary form has the value +80. This character cannot be produced on a flexowriter and is therefore used to enable the Standard Input routine to reject tapes of the wrong form.

The program being read in contains the following information:

(1) Total space required by the program (including workspace). This is stored temporarily in the first location of the program head.

(2) The program name or identifier. This must begin with a letter and is packed as 6-bit characters in the following form:



i.e. The first 3 bits are always left spare and the name is then stored so that any spare bits are at the least significant end of the locations.

Although the name must begin with a letter, the remaining characters may be taken from these detailed below.

Character	Tape Value	Packed Value
A — Z	33 — 58	1 — 26
a — z	97 — 122	27 — 52
0 — 9	16 — 25	53 — 62

N.B. Only the first six characters of any name or identifier are preserved, the remainder being ignored on input.

(3) Main entry to the program in instruction form.

(4) The checkable area pointers (where possible) in order that an internal sum check may be made on repeated entry to the program finally placed. These take the form:
00 (size of checkable area) less one: 00 (1st address of this area).

(5) The negative check sum of the tape words.

Binary Representation

Six 8-bit tape characters are used to specify each word. Of these 48 bits only 42 are actually mixed into the accumulator, the parity bit from each of the six characters having been removed. Since the machine word length is 39 bits, we have three spare bits on the 'tape word' which are used to determine how each word is modified by the basic address (i.e. first location of program) and whether it should be stored or obeyed.

These three bits are known as the 'control' and are situated at the most significant end of each word. The first two bits indicate which of the four possible modifications is required and the third bit whether the word is to be obeyed or stored (0 for obey, 1 for store).

Action Word Controls

Value

0 (00 0) R:R Add basic address to both halves and obey.

2 (01 0) R:a Add basic address to 1st half and obey.

4 (10 0) a:R Add basic address to 2nd half and obey.

6 (11 0) a:a Obey word as read.

1 (00 1) R:R Add basic address to both halves and store.

3 (01 1) R:a Add basic address to 1st half and store.

5 (10 1) a:R Add basic address to 2nd half and store.

7 (11 1) a:a Place word as read.

Functions Performed by Action Words

'Pre' action words. These come before the program proper.

(1) Clear location which holds the tape check sum.

(2) Test whether one free location exists. If not, output NOROOM. If so, set storage count of Standard Input routine to value of first free pointer. (The space (N) required by the program now printed with control 7 is thus automatically placed in the first free location.)

(3) Form the basic address modifier a : R. Use N to test whether free store large enough. Output NOROOM if not. Otherwise, reset storage count so that N is overwritten. N is also held in the program head (for later use by the 'post' action words).

(4) Form the basic address modifiers R : a and R : R.

'Post' Action Words

(1) Perform tape-sum-check. If incorrect, output ERRSUM and await next message; clear Binary flag.

(2) Form internal check-sum (if possible) and exchange with N which is held in first location of program head. Use N to adjust the first free or last free pointer.

(3) Adjust the RAP pointer and set the previous RAP pointer in the second location of the program head.

(4) Store details of the previous FF and LF pointers in the fifth location of the head.

(5) Output of the program name on the same line as the input message.

(6) If no sum check is possible, output UNCHEK on the same line as the input message.

(7) Go to await next message.

9.1. Fixed Programs

The use of fixed programs on the 503 is not recommended since they have no head and, therefore, no protection apart from a tape sum-check on input.

However, this facility has been provided for people accustomed to programming the 803 and who prefer to use TI code. When running these programs the 803 system of program management must be used.

(1) Programs written in TI code may be run on the 503 without being translated. After reading in the T2 Standard, the program is read in by typing 5. The program is entered by typing *N*. where *N* is the first instruction to be obeyed.

(2) The above method cannot be used to run programs coded into sum-checked binary by T22/23 since T23 cannot be read into the end of the store. There are two alternative methods of running this type of program:

(a) Use the 803 Operator (CON 01S, 2.2.3.1).

(b) Use the Translator T 35 to convert the sum-checked binary into a form which can be read in by the RAP standard input routine but which is placed in a fixed position. The program is input by typing IN. and entered by typing *N*. where *N* is the first instruction to be obeyed. The Translator inserts the necessary action words in the translated program.

(3) The Translator may also be used to convert programs in TI code to a form for input by RAP. The program is input by typing IN. and entered by typing *N*. where *N* is the first instruction, to be obeyed. This is an alternative method to (1).

In the case of all fixed programs, the FF, LF and RAP pointers are set as if the store is free, since fixed programs do not have a name and are therefore incompatible with the RAP system of pointers. In general it is not possible to have fixed and relocatable programs in the store together; if the fixed program is input first, it may be overwritten by the relocatable program; if the relocatable is input first, its name will be lost when the RAP pointers are reset on input of the fixed program.

9.2. Optional Placing of Programs

The Translator may also be used to prepare programs (from their TI form) which are relocatable but whose position in the store is determined by the first parameter of the input message., i.e. they are modified not according to the position of the pointers, but according to the location specified in the input message. These programs have no head and the RAP pointers will again be reset as for a clear main store.

To place a program of this type from location N onwards the user must type 'IN;*N*.' and enter by means of the instruction '*N*'.

GLOSSARY OF MNEMONIC NAMES AND THEIR ABSOLUTE EQUIVALENTS

	Data	*Location*
FF	.. .. ..	7925
LF	.. .. ..	7926
PP (RAPpointer)	..	7920
RAPword	.. ..	7914
RAPsep	.. ..	7913
IMASK	.. ..	7886
INT TR	.. ..	7887
INT PUN	.. ..	7892
INT PER	.. ..	7897

C. F. Deal *April*, 1964

Appendix 1

RAP MT : Magnetic Tape version of the Reserved Area Program

For installations with magnetic tape units this special version of the Reserved Area Program has been produced. Its aim is to improve the operating efficiency of the installation by facilitating fast retrieval of batches of programs stored on magnetic tape. This it will do in conjunction with both the STAR system (2.5.5) and the BATCH system (2.2.3.25 and 46) on which STAR is dependent.

RAP MT, when stored in place of the existing RAP, enables the user to bring to store a program batch dumped on magnetic tape by DUMP Mark 2. If X is the name assigned to the batch, the operator will type

IN; X .

in order to bring the batch to store from handler 1. No "leader" in the form of paper tape is now required.

Certain extra facilities which provide automatic transfer between STAR phases have also been included in RAP MT.

All the extra facilities are provided at the expense of some of the more seldom used RAP functions. These are listed below:

Messages not recognised

IN;N. (where N is an integer)

N. and N;S. (where N is an integer)

CANCEL.

CANCEL;"NAME".

FREE ST.

Please refer to the description of DUMP Mark 2 (2.2.3.46) for a full explanation of the facilities provided and details of the error messages which may be displayed while the batch is being input from magnetic tape.

N.B. All previous non-standard versions of RAP such as "STRAP" (STAR RAP) are superseded by RAP MT. To indicate that RAP MT is in store the figure X is output after the free store message produced when typing LIST.

2. 1 INTRODUCTION

The complete Library of programs for the 503 can be divided into three different groups: the systems programs (e.g. Algol, SAP) which are to be found in Part I of Volume 2; the applications programs (i.e. those which are job–oriented) which are available in separate pamphlets to members of the Applications Group. The remaining Library programs (machine–oriented) are described in Section 3; each program or set of related programs forms a chapter of the Section. There is a list of contents in alphabetical order. The information given in this Section refers only to this group.

Programs of any type which can be run on the 503 only via the 803 Operator using 5–hole equipment are still regarded as part of the 803 Library; however, tapes of the 803 Library programs are available on request to users of the 503. (See Appendix 1 for conversion of 803 programs to work with 503 systems programs).

2. 2 CODING SYSTEM

Each program is assigned a code name (e.g. CON 01S) consisting of up to seven letters and digits of which the first must be a letter. The first six are intended to convey the function of the program; for example, CON 01 is a converter program (the 803 Operator); the seventh is a letter specifying the type of programming system code in which it was originally written. So far three codes have been specified:–

Coding System	Code
503 Algol	A
503 Autocode	C
Symbolic Assembly Code	S
Machine Code	S

If the program is such that during the course of a run it is necessary to call it by name this is done via its code name, but otherwise the code is purely for mnemonic purposes.

2. 3 CIRCULATION OF PROGRAMS

Descriptions of all programs are issued, as soon as they become available, to every Manual holder. In general, program sheets are not supplied. The supply of

tapes is normally restricted to those customers whose equipment the program specifically concerns, i.e. tapes of a program using the line–printer are only sent to customers who have.a line–printer. The tapes provided are intended to be used only as master copies, from which copies for local use should be taken.

2. 4 CHANGES AND UPDATING

The list of contents is frequently updated and distributed together with any modifications to existing programs.

2. 5 STANDARD HEADINGS

Description of programs is normally given using the following standard headings where applicable.

(a) Code

(b) Function

(c) Store Used

(d) Method of Use
- (1) Entry points
- (2) Parameters
- (3) Error Indications
- (4) Data Tapes

(e) Format of Results

(f) Punching Instructions

(g) Configuration (in addition to the basic 503) e.g. 2 Magnetic Tape Desks.

(h) Accuracy and Time

(i) Tapes: form and whether provided or not.

(j) Process Used including references.

(k) Date of Issue and author

2. 6 ERRORS IN ISSUED PROGRAMS

Every program is carefully tested before being issued; nevertheless there is always a possibility of errors being discovered in issued programs. The 503 Librarian would welcome information on any errors discovered together with copies of the program and data tapes causing such error. Elliott Brothers cannot accept any responsibility for erroneous results produced through use of issued programs.

APPENDIX I

CONVERSION OF 803 PROGRAMS TO WORK WITH 503 SYSTEMS PROGRAMS

Most programs which can be run on the 803 can also be run directly on the 503 with the use of the 803 Operator.

Alternatively, they can be converted to use the 503 system. This appendix is mainly concerned with information on the second of these operations.

ALGOL

ALGOL programs can be converted from 803 A104 programs on the 803 by use of the program 'ALGOL 5 to 8' (see 2.2.3.5.).

The restrictions on the full ALGOL 60 language are the same as for A104.

It may, however, be necessary to change elliott orders for the following reasons:

(a) Absolute–address orders may corrupt other programs, which are allowed in store on the 503 as well as the ALGOL system.

(b) Peripheral control orders may differ.

(c) 'read' orders should take into account the normal mixing of 7 bits rather than 5 bits into the accumulator.

(d) 'punch' orders will normally send 7 bits to the output channel, not 5.

For these reasons it is recommended that ALGOL 803 A104 programs with elliott instructions should be inspected in the light of these remarks by a person well acquainted with the normal basic instructions of the 803 and the 503.

If five–hole equipment is available, the 803 Operator may be used for such programs as an interim measure.

Programs not containing elliott instructions are best used in converted form rather than via the 803 Operator.

When the current issue of 803 A104 is used with the 803 Operator it is possible to 'dump' and 'precompile'; this was not so with earlier issues.

AUTOCODE

1. Using the 803 Autocode

If five–hole equipment is available, the 5–hole programs may be translated and run using the 803 Autocode tapes (803 A3/A103) and the 803 Operator (see 2.2.3.1.).

The following notes will facilitate operation :

(i) The Reserved Area must not be protected.

(ii) The Operator must be input last, because coded tapes (sum–checked and relocatable binary) overwrite locations 8161–8192 inclusive.

(iii) The entry points are the same as for the 803, i.e. 5, 6, or 7 to translate and 16 to run the translated program.

(iv) If a translated program uses punch 3, i.e. the output writer on the 503, the output will appear unintelligible.

(v) Care must be taken that the settings on the word generator are correct. A non–zero setting left on the N2 buttons by mistake could cause the program to be entered at the reference number specified by the N2 buttons.

(vi) The machine comes to a dynamic stop when each stage of the operation is complete.

2. Using the 503 Autocode

This is the preferred method. Alterations can be carried out using EDITALL, a program in the 503 library. The program for conversion from 5 to 8 hole is ACconv S (see 2.2.3.6.).

The following notes should be helpful :

Operating

(i) 503 Autocode systems tapes are input under the control of RAP and thus the Reserved Area should be protected.

(ii) The Autocode translator is entered by typing messages on the typewriter, e.g. 7. (see also 2.1.4), to translate a program into store. The entry messages are 5., 6., 7. to translate, **16.** to run.

(iii) Check that there are no undesirable settings on the word generator left from the previous program.

(iv) Control is transferred to RAP when each stage of the operation is complete.

Points to be noted

(v) When using the 503 Autocode all input and output is in 8–hole mode.

(vi) All error indications appear on the typewriter.

(vii) The standard settings for printing results are A, 8/ and I, 4 as opposed to the settings A, 9/ and I, 12 in the 803 Autocode.

Restrictions

(viii) References to subroutines in Autocode tape 2, which may occur in a machine code block will not be automatically corrected by the conversion program. The user must alter these absolute addresses himself.

(ix) The mnemonic instruction VERIFY is always ignored by the 503 Autocode translator.

(x) 'Output n' instructions must be amended by the user.

CONVERSION OF 803 MACHINE–CODE ROUTINES FOR USE WITH 503 SAP

This process should be circumvented in a great number of cases. Owing to the far greater speed of the 503, equivalent ALGOL programs may often be produced easily and run without significant economic loss. Rewriting, or full conversion, in SAP may often easily be carried out by someone with an adequate understanding of the program. The use of EDITALL (2.2.3.21) makes these processes easier and can deal with trivial slips.

It is also possible either to repunch the program in 8–hole form or to use Autocode 5 to 8 to convert to 8-hole and then use EDITALL to carry out modifications.

However, for the sake of completeness, all possible methods are described below.

Method (A) – the quickest method

It is assumed that the 803 program is a single block and locations are expressed in relative form, i.e. in the form of an integer without a sign followed by a comma.

(a) Conversion into a SAC block

(i) Give the routine a name, differing from all other block and global names used in the program.

(ii) Punch the routine in 8–hole code, preceded by the introduction:

begin X; (where 'X' stands for the name chosen for the routine) and with the asterisk or closed bracket at the end of the tape replaced by

end X;
& (i.e. an ampersand).

(iii) If the block is a subroutine, refer to it using the instruction pair

73 X : 40 X + n

where n is the usual (relative) entry point to the block.

Note that the block must be positioned before any such reference to it, since X must be allocated before being used in a compound address such as X + n. Note also that the block cannot be entered at any of its second half instructions by the instruction pair

73 X : 44 X + n

since the assembler treats block names as if they were first half labels, and so converts

44 X + n to 40 X + n.

If a block contains second half entry points, at least one of the entry points must be introduced as a global label; this means introducing it at the head of the program and then by the side of the appropriate instruction.

This method is the quickest but leads to restrictions, i.e. a non–standard calling program, and is therefore not recommended for normal use.

(b) Conversion into a SAC program

This method can only be used if the routine's first location is its link.

(i) Give the routine a name, differing from the names of all the programs which are to use it.

(ii) Punch the routine in 8–hole code, omitting the first location, preceding it with the introductions

program X;
block X;
begin X;
and replacing the asterisk or closed bracket at the end of the tape with

end X;
trigger X;

(iii) To incorporate entry points other than the first location, label each instruction at which entry can take place with a suitable label, and introduce these labels in the block and trigger lists: so, if the labels were A, B and C we would have

block X (A, B, C);
and
trigger X, A * X, B * X, C * X;

(iv) Refer to the routine by means of the normal COMP instructions.

Method (B) – normal method

(a) Conversion into a SAC block

(i) The routine is identified and repunched as in (A) (a), but, in addition, references to its link location are replaced by references to location LINK. So, if the link location is the first in the block, an instruction like

30 0,
is replaced by
30 LINK

(ii) The content of the link location should be replaced by +0

(iii) The subroutine exits

00 0,/40 1 and 00 0,/40 n

may be replaced by

EXIT and EXIT, n

(a similar method is used if the link is other than 0,)

(iv) Instructions at which the routine is to be triggered must be identified (with labels), but it is not necessary to identify the routine's first instruction in this way. These label identifiers – call them A, B, C ... – must be introduced in the program head:

block B1, B2 ..., X (A, B, C, ...), Bn, ...;

(v) Subroutine entries to the routine are then written as

SUBR, X and
SUBR, A*X etc.

(b) Conversion into a SAC program

(i) The tape is prepared in the same way as above, (B) (a), except that LINKCP and EXITCP are used in place of LINK and EXIT, and the whole routine, in SAC block-form, is preceded by

program X;
block X (A,B,C . . .);
and followed by
trigger X, A*X, . . . ;

Method (C) – Notes on full conversion

This means that all relative addresses are replaced by identifier addresses, – otherwise the method is the same as given under (b) of Method (B).

(i) A jump instruction is changed by identifying its destination with a label, and then quoting the label in the instruction.

(ii) If possible, all other instructions in the routine should be treated as data handling instructions. This means that each location of data space is given a name; the names are introduced with a data introduction –

begin X;

data D1, D2, D3, D4, . . .;

– the original data locations are omitted from the routine, and are referred to by name instead of relative address.

Sometimes programs are written so that some of the data handling instructions are actually modifying the program, instead of working out results. The above procedure cannot be applied to such instructions.

If possible the routine should be rewritten to avoid them.

Full conversion is best carried out with a full understanding of the program, rather than by mechanical rule.

Checksums

SAC programs cannot be sumchecked if they modify themselves at runtime. 803 subroutines very often do this – either because they contain their own data space, or because the program overwrites itself. In general, then, it is better to convert an 803 subroutine into a 503 Common Program: its calling–program will be sumcheckable, even if it is not.

Another reason is that the editing of subroutines into new programs is more elegant.

Multiblock 803 routines

These must be treated more carefully: all instructions with addresses of the form 'n, m' must be replaced by instructions with addresses like 'A*B'.

In addition care must be taken with input and output routines when changing from 5–hole to 8–hole.