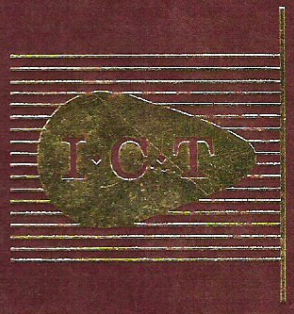
ICT 1301
TAS
Thirteen Hundred
Assembly System

Scanned 2010
at the ict1301
Resurrection Project
ict1301.co.uk

I.C.T. 1300 SERIES

TAS MANUAL



COMPUTER
PUBLICATION

Second Edition _____ December 1963

Third Edition _____ February 1965

This Edition of the TAS Manual replaces all previous Editions.

Computer Publication 3175
Printed in Great Britain by
International Computers and Tabulators Limited,
London

1300 series TAS Manual

ERRATA

Page		Correction
17	'Runout' Line 2.	" or 199 (TAS 2) lines" should read " or 235 (TAS 2) lines"
79	Last line	Last line should be, "110123 | Incorrect modification. | Abandon."
92	Problems and Solutions Paragraph 1.	The second sentence should be, "The record may also be defined as an input card, but this is not as convenient since it is required that the number of columns specified totals 80."
101	Official Subroutine numbers.	This list should be, "TAS 1 Compiler and Control: I/01/00 TAS 2 Compiler and 1" or $\frac{1}{2}$" Control: I/02/00 TAS 2 $\frac{1}{4}$" Control: I/02/01"
133	Part 2, paragraph 2 Last line	"spaces 199 lines." should be, "spaces 235 lines."
145		110303. Start procedure should be, "Press Start. (Unset M.I. 29 if TAS. 1.)"
145		110333 Start procedure should be, "Replace last card read (will be rejected if TAS 1), and press Start."

Note

Since TAS 1 "B" has been included in the Non-maintained Section of the Subroutine Library, all reference to it in the manual should be deleted. These references occur on pages 2, 37, 73, 74, 83 and 118.

I·C·T 1300 SERIES
assembly system (TAS)

CONTENTS

Appendices Page

Chapter 1

INTRODUCTION

DESCRIPTION OF TAS 1.1

This manual contains a description of **TAS.**

The user does not require a knowledge of machine code, although a general appreciation of the 1300 Series Computers - such as would be gained on an I.C.T. Computer Appreciation course - is needed.

TAS has not been designed as a full autocode, and for this reason one TAS instruction does not necessarily give rise to a large number of machine code instructions when the program is compiled. The ratio varies according to the type of TAS instruction: some, input and output instructions for instance, are very powerful; but most have been kept close to those of the machine, i.e. at a low ratio. This means that the main advantage of machine code programming - close control of the form of the program - is maintained, while the main disadvantages are eliminated.

(a) I.A.S. and magnetic drums are treated together as a single-level store. Most of the problems of program assembly are thus solved automatically.

(b) Input and output of data are reduced to their simplest forms. The TAS user has only to describe the input and output and the formats they will have. All problems of organisation of data and distribution to and from the peripheral units are dealt with automatically.

(c) All references to both data and program are mnemonic, due to the 'basic English' form of the Code language. This eliminates the possibilities of overwriting and incorrect addressing, both of which are frequent causes of programming errors when writing in machine code. Thus, the main advantage of an autocode, ease of communication between machine and user, is incorporated in TAS.

VERSIONS AVAILABLE 1.2

TAS is available in two different versions, TAS 1 and TAS 2. In usage and programmer approach only minor difference exist, TAS 2 representing an extension rather than an alteration of TAS 1.

Both versions will compile and run only on a machine of at least one full drum (12000 words) of storage.

The main differences between the versions consist in

(a) the amount of I.A.S. required and used by the object program.

TAS 1 "A" Control Pack requires and uses 400 words of I.A.S. only. Any extra I.A.S. actually present will be ignored, although it may be utilized by a programmer with a knowledge of machine coding, by means of the techniques discussed in Appendix F.

TAS 1 "B" Control Pack requires and uses 800 words of I.A.S., but some knowledge of machine coding is necessary to set up the object program pack (see Appendix S).

TAS 2, on the other hand, requires a minimum of 800 words of I.A.S. if no magnetic tape processing is to be done, or 1200 if such processing is required. At the same time, if I.A.S. over and above these requirements is available, the TAS program itself may expand into the extra area (which it will use for extra data storage).

(b) The approach to the peripheral units.

TAS 1 uses the I.C.T. PPF-C concept, whereby use of all the peripheral units is time-shared, whereas TAS 2 is based on the 'sequential' idea, whereby the use of each peripheral is time-shared with its associated data distribution routines.

TAS 2 will normally give a more efficient object program than TAS 1. However, the PPF-C concept will allow TAS 1 "B" to show to advantage when more than one peripheral is being used concurrently, and a suitable input/output ratio exists.

(c) The processing of magnetic tape.

TAS 1 takes no account of magnetic tape processing, but on a machine with 800 words of I.A.S. or more, standard tape housekeeping routines may be incorporated, as explained in Appendices F and G.

TAS 2 includes macros to handle magnetic tape on machines with a minimum of 1200 words of I.A.S.

To sum up, the main advantages of TAS 2 are that certain restrictions on TAS 1 do not apply so strictly, or at all; the utilization of I.A.S. is maximised; a much larger control routine is held in I.A.S.; and, when magnetic tape is being used, the program and data assembly and storage is automatic. These result in greatly increased object program efficiency, and will also make the program easier and faster to write.

LAYOUT OF MANUAL 1.3

The manual has been laid out so as to keep all consideration of magnetic tape processing completely separate. Thus, the considerations of data, input and output, and programming in TAS given in Chapters 2, 3 and 4 respectively, all apply to 'card machine' data processing; although all that is said here will also be relevant for users of magnetic tape.

Unless otherwise indicated, all information, rules, restrictions etc. given are common to both TAS 1 and TAS 2. Where discrepancies do exist, they are clearly indicated.

Chapter 5, 'Magnetic Tape', is divided into sections corresponding to the three divisions of the 'card machine' description. This chapter is, of course, relevant only to users with a machine of at least 1200 words of I.A.S. It may be completely ignored by all others.

Chapters 2, 3, 4 and 5 are all concerned with the actual writing of the TAS 'basic English' source program. Chapter 6, 'Using TAS', is a complete general description of the procedure to be followed from this original writing of the source program, through compilation, to object program debugging. It contains a discussion of the aids to source and object program correction and understanding given by the compiler program, and also a general description of the layout of the object program in the machine.

Finally, there is a set of Appendices, each of which deals with a specialized aspect of TAS. The reader would be well advised not to concern himself with them (unless specifically advised to do so, as he will be with Appendix H) until he has completed his study of the main bulk of the manual.

Chapter 2

DATA DESCRIPTION

TABLES 2.1

TAS allows the user to treat the 1301 as a single-level-store machine. To do this however, it needs certain information concerning data storage.

The user has control over which sections of data are to be stored on the drum, and which in I.A.S. As I.A.S. data storage is limited, lengthy or extremely infrequently used data may have to be held on the drum, in the form of TABLES.

The first information the user must give the compiler is which data is to be treated in this fashion. This he does by specifying the name and length of each Table on special sheets. This enables the compiler firstly to allocate appropriate drum storage for each Table, and secondly to create a reference list from which it may distinguish between drum-stored and I.A.S.-stored data. (Once it has been given the information with which to create this list, and has passed to the next stage of compilation, it will regard any new name encountered as that of an I.A.S.-stored FIELD.)

From this point on, throughout the program, further references may be made to any Table merely by mentioning the name it has been given, in exactly the same fashion as for any other type of data. For this reason it is not possible to use the same name both for an I.A.S. Field and for a Table, as the compiler will fail to recognise the I.A.S. Field as a different piece of data, and will treat all references to it as being references to the previously defined Table of the same name.

The format of the special sheets is shown in Appendix H. Each Table is defined on a separate line, and each line represents one Table Card in the source program. Notice that the words TABLE, NAME, ITEMS and LENGTH (preprinted on the form) appear on the first line only, and need be punched only in the first Table Card.

In the column headed 'Name of Table' is written the unique alphabetic name chosen by the user for the Table. This name must comply with the rules set out in Appendix A.

It is normal to break a Table down into smaller word groups, or records, of equal length. These are known as ITEMS, and the size of a TAS Table is defined by specifying the number of Items it contains and the length in machine words of each Item. Any Table may contain up to 999 Items, and each Item in any given Table may be up to 99 machine words in length.

Example:

Des 1 2	3	Name of 15 Table	21	Number of 27 Items	33	Length of 39 Item 44
0 1	TABLE NAME	HEADER	ITEMS	20	LENGTH	1
0 1		PAYRLL		342		7
0 1		WKSNOS		342		1
0 1						

This will cause the compiler to allocate three areas of storage on the drum, the first of 20 words, the second of 2394 (rounded up to 2400) words, and the last of 342 (rounded up to 350) words.

From now on it is possible in the program to reference any Item in a Table directly as 'TABLE a ITEM.. ...nnn', and the compiler will create a correct reference to the starting address of

FIELDS 2.2

FIELD Names are given to those pieces of data stored in the I.A.S. They must comply with the rules in Appendix A.

As mentioned above, all new names appearing once the Tables have been defined are automatically taken to be those of Fields, and therefore no separate definition of these is necessary. The compiler merely allocates I.A.S. storage to each as it encounters it, either in the Input and Output Formats described below, or in the general procedure section of the program.

A Field is allocated storage appropriate to the source program statements in which it appears. Thus if a statement reads:-

'MOVE (the contents of) 5 (words of I.A.S.) FROM A TO B'

the compiler will check that each of A and B has 5 words allocated to it, whereas had the instruction read

'MOVE 1 FROM A TO B'

then the compiler would have checked only that each Field had one word allocated to it.

The actual method of storage allocation by the compiler is somewhat involved. The user is advised to study Appendix P on this subject after making himself fully familiar with the main principles of TAS, but before actually proceeding to write programs.

The Total Storage allocated to all Fields in TAS 2 is dictated by the size of the machine, all I.A.S. remaining after a fixed allocation for program being given over to this purpose. In TAS 1 the area is not expandable but is fixed at 158 words.

In both TAS 1 and TAS 2, the area not needed for Field storage is not otherwise used.

As with Tables, Fields may be Itemized, but with an important difference: Field Items are always one word in length, and may not be defined as greater by the programmer.

Thus: if 'A' is at word 'x', then 'A Item 2' is at 'x + 1', 'A Item 3' at 'x + 2', and so on.

The contents of all Fields are at zero when the object program is first entered.

WORKING STORES 2.3

In addition to the Fields, it is possible during the procedure section (but not in Input or Output Formats) to refer to ten consecutively located I.A.S. words, known as 'WORKING STORES' and named 'WS..01' to 'WS..10'.

These stores are zeroized every time a block of program (see below, 4.2) is entered or left, and for this reason may not be used to carry information from one block to another, nor even to hold information in one block during the use of another. The only exception to this occurs when exit is made to a Global or Library Subroutine (see below, 4.6) when the state of the Stores will be preserved and restored on re-entry.

These Working Stores form an integral part of each block, and are always present, whether used or not.

CONSTANTS 2.4

Finally, TAS allows the specification of constants in the procedure section. It is in connection with arithmetic operations that constants are generally required, and it is in this type of instruction that their use is permitted.

It will be shown, as each instruction is dealt with, how one may directly reference a constant rather than a Field, Table, or Working Store, by writing its sign (+ or -) followed by the eleven digit constant required. As with Working Stores, constants, which are one machine word in length, are stored with the block of program in which they appear, TAS ensuring that no constant is stored twice with any one block. Up to 100 different constants may be used in any one block.

Although constants may be used freely, they may not be altered or updated in any way; their values remain fixed throughout the program. Should an alterable constant be required it must be set up as a Field or Working Store.

A negative reference to a constant causes its complement to be stored. Thus '+..... .....1 and '-..... .....1' are different constants. A constant of Zero may not be referenced. Should one be required, use can be made of a previously unused Working Store or Field.

In 'Sterling' instructions, only positive constants may be used. Negative sterling constants cannot be created.

Chapter 3

INPUT AND OUTPUT

Card and Line Printer Formats

For data, both Input and Output, TAS provides completely automatic distribution to and from the I.A.S. Fields. To enable the compiler to supply the control with the information necessary for this, the programmer must describe the FORMAT, or layout, of each different type of card read or punched and type of line printed. This is done by means of a series of 'keys' which are set out by the programmer on special FORMAT SHEETS (see Appendix H). Each group of five lines represents one source program Format Card.

INPUT CARDS 3.1

The fact that the format being described is that of a card to be read in, is shown by writing in the TYPE Column:

'DESn..', where n represents the designation (in the range 0 to $\overline{15}$ in TAS 2, 0 to 9 in TAS 1, punched in one column) of the particular card being described. It is standard practice for this designation to be punched in column 80 of the data card, but it may be punched in any other if so desired (see Appendix E).

Starting on the same line, in the column headed KEY, the description key of the first field of the card is written, and again, on the same line, the name chosen for that card field. This name must be that of an I.A.S. Field (not a Table or Working Store) and may not be Itemized.

Subsequent keys and names are written on subsequent consecutive lines.

The card must be described completely, starting from Column 1, and all 80 card columns must be accounted for.

At object program running time Alpha fields input are *right* justified.

Keys

Each KEY consists of five POSITIONS.

Position 1 defines the type of information within the card field

A = Alphabetic	(both zones and numerics required)
D = Denary	(numerics only)
S = Sterling	(numerics only)
I = Ignore	(irrelevant information or blank columns not to be distributed into the I.A.S. Fields. These card fields are not given names.)

Positions 2 and 3 comprise one section of the key, and show the number of card columns occupied by the Field. (The number is right justified. Four = 04 *not* 4.; 4. = 40 = Forty.)

Position 4 (preprinted on the sheets) contains "/". It should be punched only when Position 5 is used.

Position 5 is used only when the field is Alphabetic, when it must show the number of machine words needed in I.A.S. to contain the field. The TAS 2 compiler stores alphabetic information in the form 6 zones, 6 numerics, in single words. So the number of words required to store any particular Field = $\frac{n}{6}$ where n is the number of card columns occupied by the Field. The result must, obviously, be rounded up to the next whole number if necessary. TAS 1 stores alphabetic information 12Z, 12N, in pairs of words; and so the result obtained from the above formula must be rounded up, if necessary, to the next whole *even* number.

Since it is possible to describe only five card fields on each TAS Format Card, it is necessary to repeat the information in the Type column at every fifth key. Each new Input Card description must commence a new TAS Format Card.

A Sterling or Denary card field may not exceed eleven card columns in length (one I.A.S. word when distributed), nor may an Alphabetic field exceed 60 columns (10 I.A.S. words) in TAS 2, or 48 (8 I.A.S. words) in TAS 1. A maximum of 40 (TAS 2) or 30 (TAS 1) fields may be described for any one input card format.

Example:

DES 1 2	TYPE 3 8	KEY	FIELD NAME	CARD No. COLS 69-74	
03	DES 11	A10/2	NAME		Name of part
		I06/			
		D10/	REF NO		Reference Number
		S11/	PRICE		Price of part
		I17/			
03	DES 11	D11/	STOCK		Current Stock
		D08/	ORDLEV		Re-order level
		I07/			
		/			
		/			

The designation of the card may be 'Ignored' in distribution, even if its value is to be tested, since the testing is an optional facility of the READ verb (see 4.7.14), on which the distribution routines generated by these formats have no bearing.

It is not necessary to specify Input Formats, if no cards are to be read in.

OUTPUT CARDS AND LINES 3.2

For these the Type column contains either 'LINEnn' or 'CARDnn' as appropriate, where nn is the code number allocated by the programmer to the particular format being described. When printing or punching, the particular format desired to be output is then specified. It should be remembered that the number of a CARD format need have no connection with the designation actually punched into that card (which will be described as a field). It is merely for referencing the format description.

Every format number must be unique: a CARD and a LINE may not have the same number. LINE formats, if any, should precede those of CARDS.

In TAS 2, up to thirty different output formats may be described ($1 \leqslant nn \leqslant 30$). Like Input Formats, these may be presented to the compiler in any order, and the code numbers need form no consecutive sequence.

TAS 1, on the other hand, has a limit of 20 different formats, which must be presented to the compiler in format number order. These numbers must start at 01 and rise in increments of one. Every TAS 1 source program must contain at least one output format.

As in the case of input cards, the fields on each different line to be printed, or card to be punched, must be described by keys. These keys are similar to those of the input formats in type, but are *not* identical in use. For instance, it is necessary to describe only those fields which are to be printed or punched, TAS arranging for the remainder of the line or card to be left blank. Names allocated to fields must again be those of I.A.S. Fields, and not Itemized.

Keys

Position 1 Only three different keys are needed to define the types of information output, viz:

A = Alphabetic

D = Denary

S = Sterling

Positions 2 and 3 represent the card column or position in the line of print in which the least significant character of a field is to appear. Position 2 may vary between 0 and $\overline{12}$ written in a single column and position 3 between 0 and 9, thus allowing all 120 positions on a line to be defined. Print positions are numbered 1 to 120 from left to right across the printed page, Card columns from 1 to 80 across the card.

Position 4 (preprinted) contains "/", but again should be punched only when Position 5 is present.

Position 5 has a variety of uses depending upon the type of key in Position 1.

When Position 1 = A (Alphabetic Type), Position 5 represents the number of machine words occupied by the field, as in the case of input format keys. (Again the maximum is $\overline{10}$, representing 60 characters.) The data will be accepted as being in the form 6Z, 6N for TAS 2, or 12Z, 12N for TAS 1; alpha constants must be in the appropriate form.

When Position 1 = D (Denary Type), Position 5 represents the number of digits after the decimal point (this is used only in LINE formats as no decimal point is ever punched). If its value is non-zero, a decimal point is automatically inserted into the line of print, and the integers are shifted one place to the left; no decimal point will be printed if the position is zero or blank.

Finally, when Position 1 = S (Sterling Type), Position 5 specifies the number of decimal places of pence. If the value is non-zero, a decimal point is automatically inserted into the line of print, and the integers are shifted one place to the left.

There is, of course, no restriction on punching out denary or sterling fields which contain decimal places, but no special action on them will be taken. They will merely be output digit for digit. In all cases, a printed sterling value will be expanded to the left with a single space between the pounds and shillings and between the shillings and pence; no '£' sign is printed.

e.g. The sterling constant 000, 213, 095, 142 is printed:

213 9 5.142 if Position 5 is equal to 3

and 213095 14 2 if Position 5 is zero or blank.

If a printed sterling value should be less than one shilling, then a zero will be printed in the digit shilling position. Hence, if it is known that a certain sterling field will never be more than 11d, and suppression of the zero in the shilling position is required, the field should be defined and treated as an alphabetic one.

In all cases of denary and sterling values, non-significant zeros are suppressed to the unit integer. The complement of any negative number presented is automatically output, but its sign is not created. There is, of course, nothing to prevent the TAS user from creating the sign himself, provided it is allowed for in the format description as a separate character.

Example:

DES 1 2	TYPE 3 8	KEY	FIELD NAME	CARD No. COLS 69-74	
03	LINE04	A10/2	NAME		Name of part
		D26/	REF NO		Reference number
		S37/	PRICE		Price of part
		D66/	STOCK		Current Stock
		D74/	ORDLEV		Re-order Level
03	LINE04	D116/3	AVGE		Average (3 decimal places)
		/			

(Notice that the first five fields described are identical as to name, type, *and position* with those in the 'Input' example given above.)

In TAS 2, any particular I.A.S. Field may be output any number of times in any output format. The Fields may be defined in any order, since their relative positions as output are determined by the 2nd and 3rd Positions of their respective keys. A maximum of 29 fields may be described in any one output format.

In TAS 1, any particular I.A.S. Field may be output only once in any one format (but in any number of different formats). The order of the keys is also significant. Fields in any one output format must appear in the order in which they have already been presented to the compiler, in Input formats, or in previously described Output formats. A maximum of 15 fields may be described in any one output format.

Chapter 4

PROCEDURE

The program itself is written on program sheets (see Appendix H) in the form of a series of different TAS 'verbs' or orders, each of which, together with its object and subsidiary key-words and their objects, forms an instruction or macro.

In the separate sections on each verb below, the Formats are set out, and variations are dealt with. All data names of which 'Itemization' is permissible are marked with an asterisk. The name of a Field, Table, or Working Store is signified by the word 'Name'.

PROGRAM SHEET LAYOUT 4.1

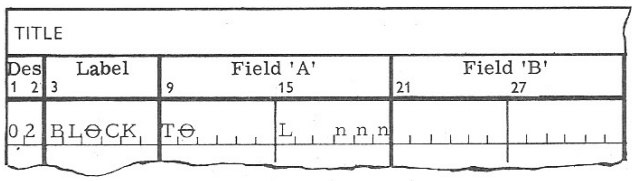
TAS words like TAS Data Area names have a maximum length of six characters, and the program sheet has a layout identical to that of the TAS program card. Each line on the sheet may hold one TAS instruction only, and represents one TAS source program card. Each card field is split into two portions, representing generally the operator and operand of an instruction or part of an instruction.

BLOCKS 4.2

TAS programs are written in blocks, or groups of related instructions, as are machine-coded programs. It is the user's responsibility to divide his program into blocks, sized so that the compiled instructions and their associated constants and Working Stores will not occupy more than 200 machine words. 50 TAS instructions will usually fit easily into a block, but many more may do so, depending on the type of instruction.

It is safer to make the blocks comparatively small for an initial compilation, and then extend them by amalgamation when their sizes are known from the information printed out by the compiler.

Each block of program must be headed by a 'Block Heading Card' defining the limits of that block:



where 'L nnn' is the label of the last TAS instruction of the block.

Note: TAS does not transfer control automatically from the last instruction of a block to the first of the next. The last instruction of each block must therefore be itself a control transfer instruction of some type, either to another block or back into the present one.

LABELS 4.3

Instructions in TAS are not sequentially numbered, as they are in machine code.

Any instruction which is to be referred to elsewhere in the program, either as the object of a control transfer or of modification, must therefore be labelled. A label is written in the 'Label' column of the program sheet, as 'L nnn', where nnn is the number of the label in question. Label numbers may range between 0 and 999, may appear randomly (and need not form a consecutive sequence even if listed in ascending order), but each must be unique. Label 0 must be used to represent the initial entry point of the object program, but need not be the first label of the first block.

The last TAS instruction of each block, since it is referenced on the 'Block Heading Card', must also be labelled. In addition, the final instruction of a subroutine must be given the special label 'ENDSUB'. This will be more fully dealt with in the section on Subroutines.

STERLING 4.4

As the 1301 is capable of operating directly in sterling, facilities have been provided in TAS to retain this feature with those verbs to which it is appropriate.

To qualify an instruction so that sterling rather than decimal arithmetic will take place, it is necessary only to place a '£' sign in position 5 of card field 'A'.

It is also possible to set the Sterling Position Register of the machine (which controls the 10/- position during sterling arithmetic) if so desired, before entering a sterling instruction. This is done by following the '£' sign described above by the 10/- position desired.

e.g. 'LOAD£$\overline{10}$' etc .... will cause sterling arithmetic in the instruction, with the '10/-' in digit 10 (i.e. no decimal places of pence), whereas

'SUM . £6' etc .... will cause sterling arithmetic with the '10/-' in digit 6 (i.e. four decimal places of pence).

It is possible to set the Sterling Position Register to any digit between 2 and 12. It is not possible to set it to 1, and setting 0 will have no effect, being ignored by the compiler as if the position had been left blank.

Once set to a particular place, the sterling position register remains fixed until reset. The only TAS Instructions which may reset the register, without a specific instruction to do so being given by the programmer, are the 'PRINT' and 'PUNCH' verbs. Care should be taken to reset the register after a 'PRINT' or 'PUNCH' instruction before performing sterling arithmetic.

'PRES' 4.5

'PRES' is the name given in TAS to the accumulator of a 1300 Series machine i.e., to Register B. It may be referenced in many (but not all) instructions in place of the name of a data storage area. Thus, while 'CLEAR VALUE' would zeroize the data area 'VALUE', 'CLEAR PRES' would zeroize the accumulator. This will be made clearer when certain of the verbs themselves and their formats are discussed; as will also the restrictions on the referencing of 'PRES'.

SUBROUTINES 4.6

It may become necessary during the flow charting of a program to make one particular section or routine available to various parts of the whole routine. This is done by treating the section as a 'subroutine' which, when control is transferred to it, will store a 'link', through which it will return finally to the next sequential instruction of the main program.

Subroutines in TAS have control transferred to them by means of the 'OBEY' verb, and discussion of them in detail will be delayed until we come to consider this verb.

VERBS AND INSTRUCTIONS 4.7

The verbs described in this section fall into five groups according to their uses:

(a) Program Control and Organisation:

STOP, GO TO, COMPare, TEST, and TURN.

(b) Calculation:

CLEAR, LOAD, SUM, MULTiply, and DIVide. All perform single-word (eleven-digit) length arithmetic, and all except SUM affect only the single data *word* specified in each card field.

(c) Data Control and Organisation:

PACK, MASK, SHIFT, and MOVE. Again all except MOVE affect only the single word referenced in each card field.

(d) Input and Output Macros:

READ, PRINT, and PUNCH.

(e) 'Facility' Macros:

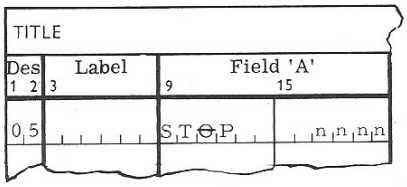
MODIFY, 1301, OBEY, and RENAME.

(The sixth group, the Magnetic Tape Macros, applicable to TAS 2 only, are discussed in Chapter 5).

STOP 4.7.1

This is inserted at any point in the program at which it is required to stop the computer.

The basic format is

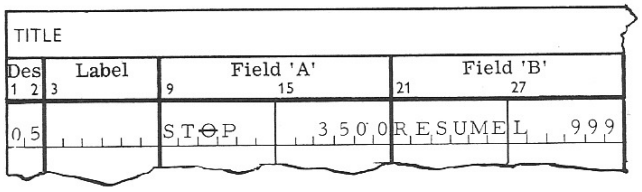


The number 'nnnn' will be displayed in Control Register 3 when the computer has stopped. nnnn must be within the limits 2001 to 3999.

Resume

If it is required to restart, after a stop, at an instruction other than the next in sequence, the keyword 'RESUME' must be entered in the first half of field B. When Start is pressed, control will be transferred at once to the label specified in the second half of field B.

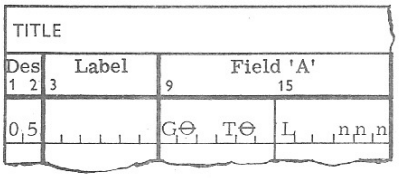
Example



A *numbered label* must be used. It is not possible to resume at the special label 'ENDSUB'.

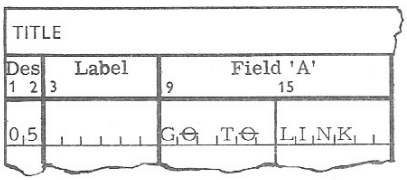
GO TO 4.7.2

Instructions are normally obeyed in physical sequence. This verb is used to transfer control unconditionally from one point in the program to another not in sequence.



Within a subroutine the special label 'ENDSUB' is an acceptable alternative to a numbered label.

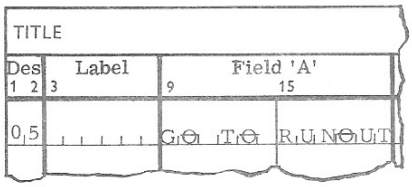
Also within a subroutine the special form



may be used to cause an immediate exit from the subroutine.

Runout

There is also a special form of this verb, best thought of as a completely separate verb. It is



and causes the print buffer to be emptied, and 200 (TAS 1) or 199 (TAS 2) lines of paper to be spaced. This instruction *must* appear at least once in any program in which printing has occurred, between the last print instruction and the final stop. It is advisable to make it the last instruction before the final stop, to avoid errors arising from additional output instructions being inserted later.

The keyword 'RUNOUT' may appear only in this verb and in this fixed format. It is *not* a label, and may not be used as such.

COMPare 4.7.3

This verb is used to transfer control to different points in a program, conditional upon different results of a comparison between two quantities.

The standard format is

Field 'A' 9–15		Field 'B' 21–27		Field 'C' 33–39		Field 'D' 45–51		Field 'E' 57–63	
COMP.	name A	WITH	name B	MORE	L nnn	EQUAL	L ppp	LESS	L qqq
	PRES								

(When the COMP. occurs within a subroutine, 'LINK' and 'ENDSUB' are acceptable alternatives to the numbered labels shown in the format.)

If the data to be compared are sterling, the '£' sign must be inserted, and also the 10/- position if necessary (see 4.4 above). 'PRES' is an acceptable alternative to name A, but not to name B.

It is possible to compare directly with any constant other than Zero. In this case the format becomes

Des 1–2	Label 3	Field 'A' 9–15		Field 'B' 21–27		Field 'C' 33–39		Field 45
05		COMP.	name A	+/- nnnnn	nnnnnn	MORE	L nnn	EQUAL
			PRES					

Subject to these minor variations, the format is fixed. The three conditions MORE, EQUAL, LESS must always appear with their associated labels. One may not be omitted because it is of no interest to the programmer, nor even because it is an impossibility that it should ever occur.

Control will be transferred, on any particular occasion, to the destination specified with the condition that is true on that occasion. Notice that the conditions refer to *name A* as compared with name B.

Any two, but not all three, of the specified destinations may be identical. Should one of the destinations be the next instruction in sequence, then that instruction must be labelled and specified in the normal way.

Modification

Data area names in this instruction may not be modified under any circumstances.

TEST 4.7.4

This verb is used to transfer control dependent on the state of one of the indicators available to the programmer. These are the nine program indicators 10-18 (the state of which the programmer may himself control by means of the 'TURN' verb); the nine manual indicators 20-28, over which the programmer also has control; and the three 'mill' indicators 'POS' (positive), 'NEG' (negative), and 'ZERO', which are turned on and off automatically according to the result of each arithmetic operation as it passes through the mill. (It is also possible to test 'OVERFL' (Overflow), but see the note below).

The format is:

TITLE							
Des 1 2	Label 3	Field 'A' 9	15	Field 'B' 21	27	Field 'C' 33	39
05		TEST	nn	SET	Lnnn	UNSET	Lnnn
			POS		LINK		LINK
			NEG		ENDSUB		ENDSUB
			ZERO				
			OVERFL				

Note that control may be transferred to LINK or ENDSUB only within a subroutine.

The Test for 'SET' must always be present, but the Test for 'UNSET' in card field C may be omitted, in which case the program will continue to the next instruction in sequence on those occasions when the indicator is not set.

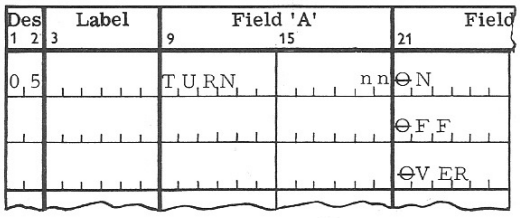
Note:

'OVERFL' becomes set when the result of an arithmetic operation exceeds eleven digits. However, although it is turned on automatically, like 'POS', 'NEG', and 'ZERO', it can only be turned off by testing. For this reason, it is necessary to 'TEST OVERFL' both before and after any operation for which this condition is significant.

TURN 4.7.5

The programmer may control the state of the nine 'program indicators' numbered 10 to 18. This verb is used for the purpose.

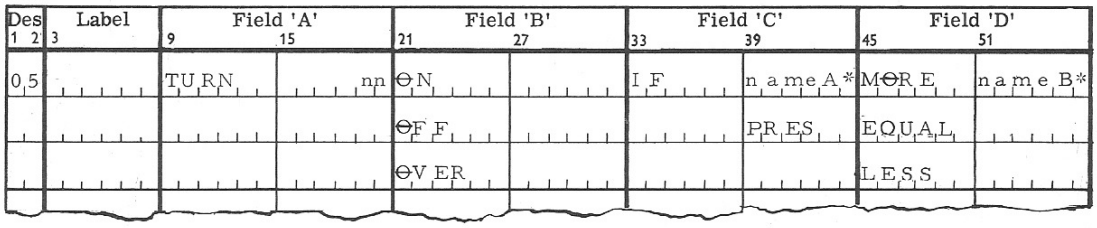
The basic format is



where 'nn' is the number of the indicator in question. 'TURN nn OVER' will turn 'nn' on if previously off, and off if previously on.

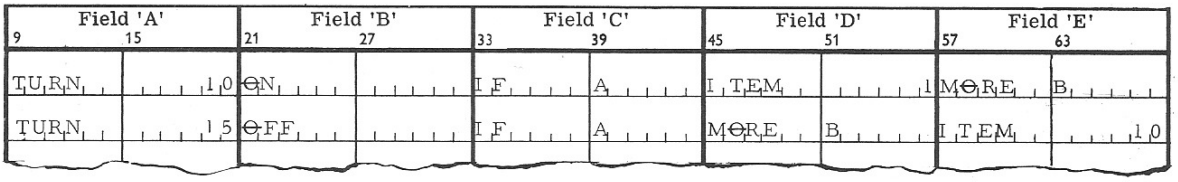
(Notice that the words 'SET' and 'UNSET' are *in no way* connected with this verb. They are keywords of the verb 'TEST'.)

This alteration in state can be made conditional on the truth of a proposition. In this case the format is extended to

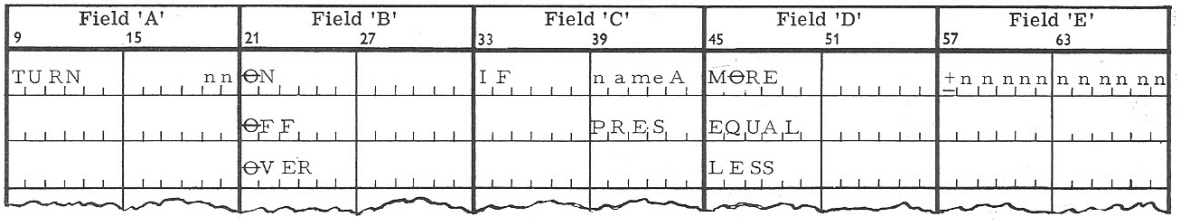


PRES is an acceptable alternative to name A, but not to name B.

Either, but not both, names may be Itemized e.g.:



It is possible to make a direct comparison with a constant, but in this case there can be no Itemization. The format is



As with the COMPare function, it is *name A* which is referred to as being MORE, EQUAL, or LESS than the name or constant with which it is compared.

Sterling

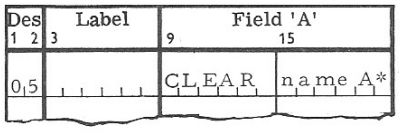
If the quantities compared be sterling, the '£' sign, and the 10/- position, if necessary, are inserted in the appropriate position.

Modification

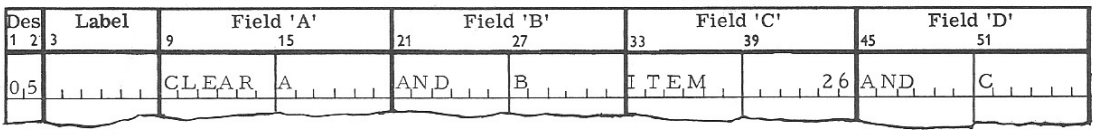
Data area names in this instruction may be modified.

CLEAR 4.7.6

This verb zeroizes the *first word only* of the area specified.

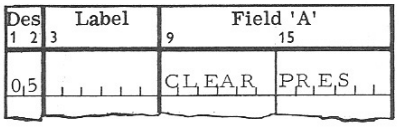


By means of the subsidiary keyword 'AND' several words may be cleared in one instruction, e.g.



Note: To clear large areas it is usually simpler to enter '1301' (see 4.7.19 below) and then use 'chaining', which is not possible in TAS.

Register B may be zeroized by means of the instruction



and data areas may be cleared in the same instruction by use of the keyword 'AND'; but PRES may appear only in field A, as the main object of the verb.

Modification

'CLEAR' may be operated upon by the verb ' MODIFY', but

(a) No more than four names in one instruction may be modified.

(b) A modified Table Name may not occur in the same CLEAR instruction as one which is not modified.

LOAD 4.7.7

This is the basic verb for performing addition and subtraction. Both denary and sterling quantities can be dealt with. The verb differs from most others in that the format is completely variable.

The effect of the LOAD verb itself is to place in the accumulator the quantity named in the second half of field A. Operations may then be performed with or upon this quantity using the following keywords:

(a) ***Add*** The contents of the data word named in the second half of the field containing this key are added to the quantity already in the accumulator.

(b) ***Sub*** Similar to (a) but the contents of the named data word are subtracted.

(c) ***+ or –*** The constant specified in the following 11 digits which appear across the field containing one of these signs is added to or subtracted from the quantity in the accumulator.

(d) ***Add To*** The quantity at the time in the accumulator is added to the data word named in the second half of the field.

(e) ***Sub Fr*** Similar to (d), but the quantity is subtracted from the data word named.

(f) ***Result*** The quantity in the accumulator at the time is transferred to the data word named.

These secondary keywords, with their associated names, may appear in any order across the whole instruction, except that the last keyword appearing in any LOAD instruction must be ADD TO, SUB FR, or RESULT. This is *not* to say that any one or more of these key words may not appear anywhere else in the instruction.

Example

Field 'A' 9–15		Field 'B' 21–27		Field 'C' 33–39		Field 'D' 45–51		Field 'E' 57–63	
LOAD	A	ADD	B	ADD TO	C	SUB	D	RESULT	PRES
LOAD	PRES	+	20	SUB FR	E	–	10	RESULT	F

If A=10, B=2, C=3, D=4 and E=50, then the following will occur:

Function		contents of -						
		'PRES'	A	B	C	D	E	F
Load	A	10	10	2	3	4	50	n
Add	B	12	10	2	3	4	50	n
Add To	C	12	10	2	15	4	50	n
Sub	D	8	10	2	15	4	50	n
+	20	28	10	2	15	4	50	n
Sub Fr	E	28	10	2	15	4	22	n
–	10	18	10	2	15	4	22	n
Result	F	18	10	2	15	4	22	18

The figures underlined are those which are affected by each Function. Notice how certain keywords affect PRES but not the data word, while others affect the data word but not PRES. *No one keyword affects both.* Notice also that, as '+..... ....20' would not have been a legitimate last Function, it was possible to say 'RESULT PRES' and then 'LOAD PRES', and so continue over one instruction. As neither affects the data in any way, the compiler will not actually generate any machine instructions for these two orders. Finally, since the result of a LOAD instruction is always left in PRES, it is possible to continue a long series of calculations, by commencing each new instruction with 'LOAD PRES'.

Itemization

Any name in a Load verb may be itemized:

Field 'A' (9–15)		Field 'B' (21–27)		Field 'C' (33–39)		Field 'D' (45–51)		Field 'E' (57–63)	
LOAD	X	ITEM	27	ADD	Z	ITEM	7	RESULTY	
LOAD	PRES	ADD TOW		ITEM	19	ADD	WS 04	ADD TOQ	

Mill Indicators

LOAD and RESULT do *not* affect the mill indicators. All other keywords do, so it is generally possible to discover the sign of the result of an arithmetic operation by using the verb 'TEST', immediately following the LOAD instruction, to test one of the mill indicators.

Special Forms

(a) Sometimes it is desired to discover the sign of a stored quantity. The simplest way of doing this is to call it or its complement to the accumulator in such a way as to affect the mill indicators. (That is, to perform a 'clear add' or 'clear subtract'.) To do this a special form of the LOAD verb may be used:

Des (1 2)	Label (3)	Field 'A' (9–15)		Field 'B' (21–27)	
0,5		LOAD		ADD	name A*
				SUB	

(b) To bring a constant into the accumulator initially, another special form of the verb is used

Des (1 2)	Label (3)	Field 'A' (9–15)		Field 'B' (21–27)	
0,5		LOAD		+/- nnnnn	nnnnnn

(Despite the similarity to the first special form, this does not affect the mill indicators.)

Both these special forms may then continue across fields C, D, and E in the normal fashion.

(c) Finally it is possible to increase or decrease the value of a given I.A.S. Field or Working Store (but not of a Table or PRES) by 1, 2, or 3, by means of a 'condensed' LOAD instruction:

Des (1 2)	Label (3)	Field 'A' (9–15)		Field 'B' (21–27)	
0,5		LOAD	n	ADD TO	name A*
				SUB FR	

where $1 \leq n \leq 3$. This special format is fixed, and no further keywords may follow. Although it has been designed for use in counting, there is no formal restriction upon its use.

This form does not affect PRES in any way.

Sterling

Sterling arithmetic may be performed in all formats of the LOAD verb, by inserting the '£' sign in the normal fashion. In a 'condensed' LOAD, the Sterling Position Register setting *must* be $\overline{10}$.

PRES

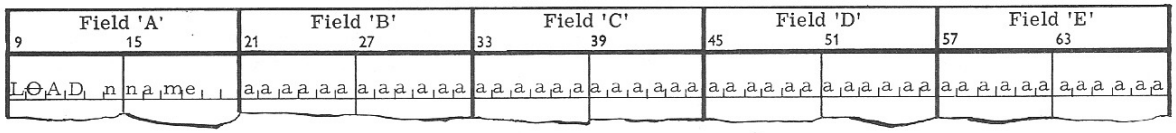
PRES may be the object only of the 'LOAD' verb itself, and of the keyword 'RESULT'. It may not be used after ADD, SUB, ADD TO, or SUB FR.

Modification

Up to four data word names may be modified in any one 'LOAD' instruction.

LOAD Field with literals (TAS 2 only)

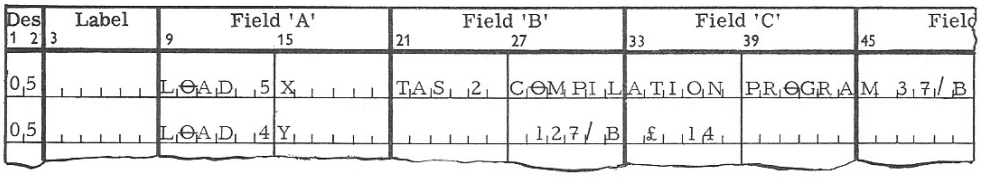
This facility is entirely different from all other forms of the LOAD verb in both concept and function. The format is:



where the name specified may only be that of an I.A.S. Field, and may not be Itemized.

It fills the first 'n' words of the Field with the alphanumeric literals set out in card fields B, C, D and E. Each half card field represents one word (of six alphanumeric characters) of the I.A.S. Field and 'n' may lie in the range 1 to 8.

Examples:



In the second example above, the first and fourth words of I.A.S. Field 'Y' will be loaded with zeros.

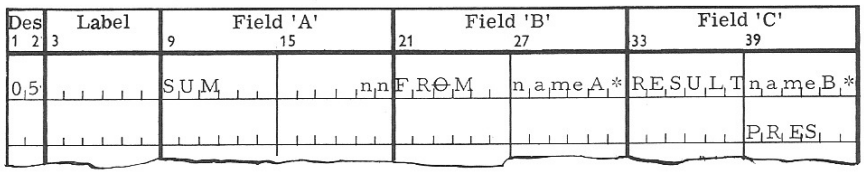
By means of this facility the user may create alphanumeric constants as required, thus avoiding the need for some Tables. All heading lines, for example, could be defined as being of the same format, and the variable information loaded into the defined I.A.S. Fields as necessary, before each PRINT instruction. However, the user is warned that each instruction of this type will create 'n' constants in the object program block. For this reason discretion is urged in its use.

This instruction affects neither PRES nor the mill indicators. It may not be modified in any way.

SUM 4.7.8

This verb will total up to 50 consecutive words of storage, placing the resultant grand total in the location specified.

The fixed format is:



where 'nn' is the number of words to be added together ($2 \leqslant nn \leqslant 50$), and name A is the first of the consecutive locations. Either or both location names may be itemized.

Sterling

The verb may operate in sterling, but care should be taken to ensure that the 10/- position of all words to be summed is the same, and that denary and sterling quantities are not dealt with by the same instruction.

PRES

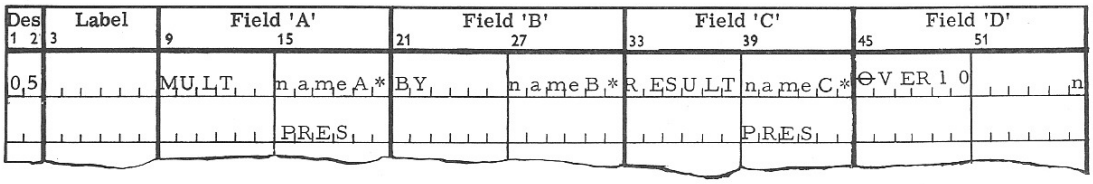
The result may be held in PRES, but obviously PRES may not be specified in card field B.

Modification

This verb may not be 'modified'.

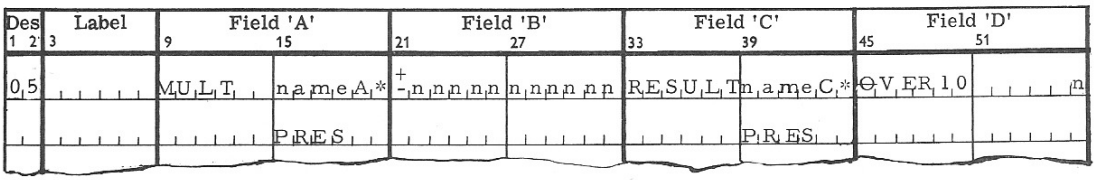
MULTiply 4.7.9

This instruction has the format:



Only *one* name may be itemized in any one instruction.

The multiplier may be a literal (or constant), in which case the format is slightly altered to



It is possible to divide automatically the product of a multiplication by a selected power of 10. For this reason the user must specify, after the keyword 'OVER10', the power of 10 by which he wants his product divided. Should he not want this facility, 'OVER10' *must still be shown* , and 'n' must be set to zero.

Sterling

Sterling multiplication may be performed by inserting the '£' sign (and the 10/- setting if necessary). The sterling quantity must be that specified in the second half of card field A, i.e. the multiplicand. The multiplier must always be denary.

The result of a sterling multiplication will always contain the same number of decimal places of pence as did the multiplicand.

PRES

The multiplicand and the result, but not the multiplier, may be held in PRES.

Modification

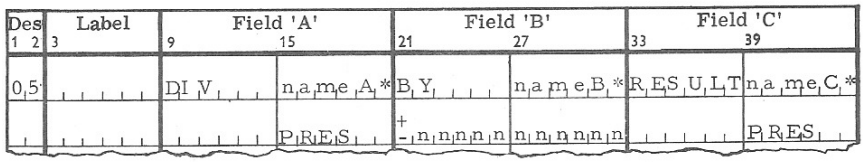
Data word names in this instruction may be modified.

DIVide 4.7.10

The format of this verb is partly variable according to the type of result which is required.

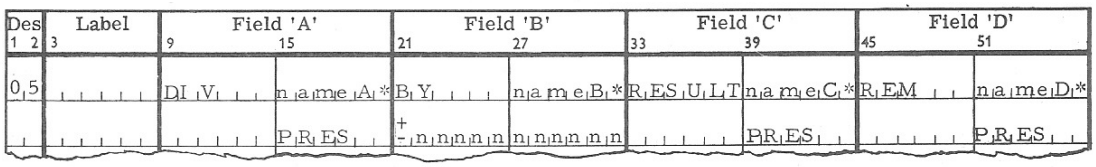
It is possible to select:

(a) An unrounded quotient, and no remainder (basic format):



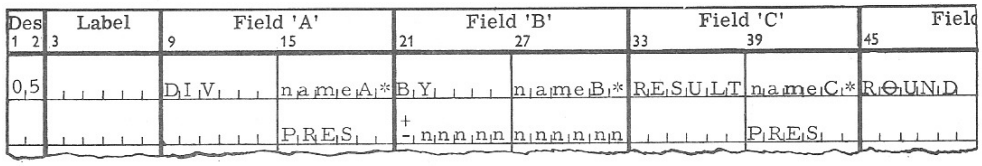
Any *two* names may be itemized. Notice that the divisor may be expressed as a constant, but may not be PRES.

(b) An unrounded quotient, *and* a remainder:



Any *one* name may be itemized. It is not possible to direct RESULT and REM to the same location. The remainder will always be correctly signed, and, in the case of a sterling dividend. will be left as an amount of £.s.d.

(c) A rounded quotient:



Any one name may be itemized. The result is rounded to the nearest whole number (or *up* if the remainder is exactly half the divisor). The remainder is not available.

Limitation

In a division neither the dividend nor the divisor may exceed eleven digits.

Sterling

A division may be decimal by decimal, sterling by sterling or sterling by decimal, but, in the case of sterling quantities, the 10/- must always be in digit position 10. Similarly, denary quantities are always regarded as integers, so that any scaling required must be carried out by program prior to obeying the division instruction itself.

Notice that, while it is possible to describe the divisor as a positive or negative constant in the normal manner, this constant is not permitted to be sterling. Finally, if the divisor is sterling, Field B, as well as Field A, must carry a £ sign in position 5.

Modification

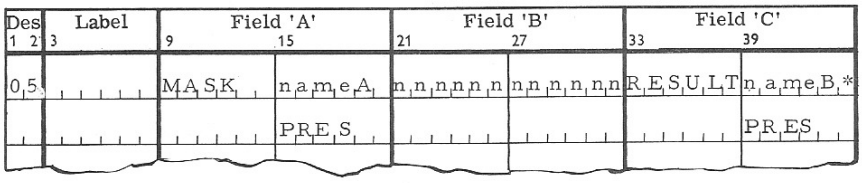
Data word names in this instruction may be modified.

Note: While MULTiplication is performed by hardware in the 1300 series computers, DIVision is not, and requires a subroutine. In addition, in TAS, DIVision requires drum transfers. For these reasons multiplication is considerably faster than division, and should always be selected when a choice between methods exists.

MASK 4.7.11

This verb performs a 'logical and' whereby only those bits existing in both original data word and masking constant are present in the final result.

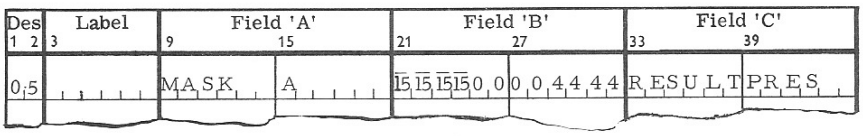
The fixed format is:



The masking constant must be expressed as a literal, which may have any value between 1 and $\overline{151515151515}$ $\overline{151515151515}$. This masking constant is the only exception to the general rule that in TAS a constant may have a maximum length of eleven digits plus sign.

Example

A: 000123 000456



A: unchanged

PRES: 000100 000444

This instruction affects the three mill indicators, POS, NEG and ZERO, which may then be tested for the sign of the result. OVERFL is not affected.

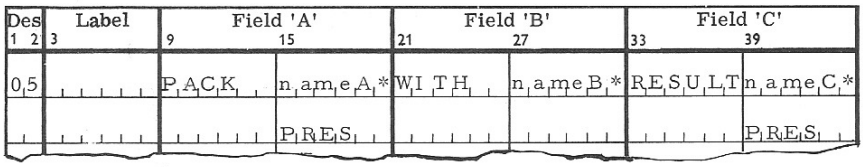
Modification

In a MASK instruction, only I.A.S. Field names may be modified.

PACK 4.7.12

This verb performs a 'logical or', whereby any bit existing in *either* original data word is present in the result.

The fixed format is:



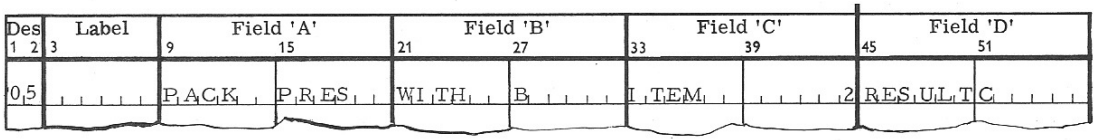
The result of the operation is left in both PRES and name C. Name A and name B are unchanged.

Any *two* names may be Itemized in any one instruction.

Example:

PRES = 000000 300456

B ITEM 2 = 123000 $\overline{1}$000000



PRES = 123000 $\overline{1}\overline{1}$00456

B ITEM 2 = 123000 $\overline{1}$000000

C = 123000 $\overline{1}\overline{1}$00456

PRES

PRES may *not* be specified in place of name B, i.e. after 'WITH'.

Modification

In a PACK instruction, name A and name B may be modified only if they are I.A.S. Field names; name C may not be modified. No name may be modified in an instruction beginning 'PACK PRES'.

SHIFT 4.7.13

This causes the digits within a word to be rearranged.

The format is:

Des 1 2	Label 3	Field 'A' 9 15		Field 'B' 21 27		Field 'C' 33 39	
0,5		SHIFT	name A*	LEFT.C	nn	RESULT	name B*
			PRES	LEFT.			PRES
				RIGHTZ			
				RIGHT.			

where $1 \leqslant nn \leqslant 12$.

The data word on which the operation is to take place is specified in the second half of card field A. In the first half of the next field is specified, by means of a keyword, the type of rearrangement desired. These keywords, and their effects, are:

(a) LEFT.C ('Left Circulate')

The digits move to the left. As a digit leaves the most significant end of the word it reappears in the least significant position, as if the word were an endless chain of twelve links.

(b) LEFT..

The digits again move left, but as each leaves the most significant end of the word it vanishes. Zeros enter the vacant positions at the least significant end of the word.

(c) RIGHTZ ('Right Zeroize')

Similar to LEFT.., except that the movement is to the right.

(d) RIGHT.

The digits again move right but if the original number is negative, the result will be made negative by 9's rather than zeros entering at the most significant end.

In the second half of the same card field as the keyword is specified the number of places the digits are to be moved.

Finally, the keyword 'RESULT' is inserted, followed by the location in which the result is to be stored.

The contents of name A are not affected by this verb (unless name B ≡ name A).

PRES

Either or both of name A and name B may be replaced by 'PRES'. In any case 'PRES' will always hold the result.

Modification

Data area names in this instruction may be modified.

Example

A = 012345 6789$\overline{10}\overline{11}$

FUNCTIONS						RESULTS		
Field 'A' 9–15		Field 'B' 21–27		Field 'C' 33–39		Location	Affected	Contents
SHIFT	A	LEFT C	4	RESULT B		PRES	Yes	4 5 6 7 8 9 $\overline{10}\overline{11}$ 0 1 2 3
						A	No	0 1 2 3 4 5 6 7 8 9 $\overline{10}\overline{11}$
						B	Yes	4 5 6 7 8 9 $\overline{10}\overline{11}$ 0 1 2 3
SHIFT	A	LEFT	6	RESULT A		PRES	Yes	6 7 8 9 $\overline{10}\overline{11}$ 0 0 0 0 0 0
						A	Yes	6 7 8 9 $\overline{10}\overline{11}$ 0 0 0 0 0 0
						B	No	4 5 6 7 8 9 $\overline{10}\overline{11}$ 0 1 2 3
SHIFT	PRES	RIGHT	1	RESULT B		PRES	Yes	9 6 7 8 9 $\overline{10}$ $\overline{11}$ 0 0 0 0 0
						A	No	6 7 8 9 $\overline{10}\overline{11}$ 0 0 0 0 0 0
						B	Yes	9 6 7 8 9 $\overline{10}$ $\overline{11}$ 0 0 0 0 0
SHIFT	PRES	RIGHTZ	1,1	RESULT PRES		PRES	Yes	0 0 0 0 0 0 0 0 0 0 0 9
						A	No	6 7 8 9 $\overline{10}\overline{11}$ 0 0 0 0 0 0
						B	No	9 6 7 8 9 $\overline{10}$ $\overline{11}$ 0 0 0 0 0

MOVE 4.7.14

This instruction is used to transfer the contents of an area, or group of successive areas, to another area or group of areas. Both areas retain their original identities and the contents of the first are unchanged.

The format (which is fixed) is:

Des 1 2	Label 3	Field 'A' 9 15		Field 'B' 21 27		Field 'C' 33 39	
0,5		MOVE	nn	FROM	name A *	TO	name B *

where nn = the number of *machine words* (up to 50) to be moved. Either or both names may be Itemized.

Note:

(a) This order performs an absolute one-for-one data transfer. The familiar machine code "chaining" facility is not available.

(b) If both names are those of Tables, then only *one* word may be moved. In any case, it is always preferable to perform a move of this nature in 2 stages through an I.A.S. Field or Working Store.

PRES

PRES may not be referred to at all in this verb.

Modification

This verb may be modified, subject to the following restrictions:

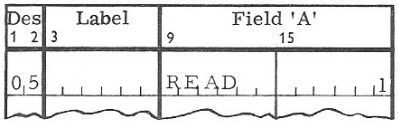
(a) Only 20 words in TAS 2, or 10 in TAS 1, may be moved.

(b) If Tables are referred to in the transfer only one address may be modified.

READ 4.7.15

This verb is the input instruction, and may also specify the action to be taken appropriate to the designation of a card read.

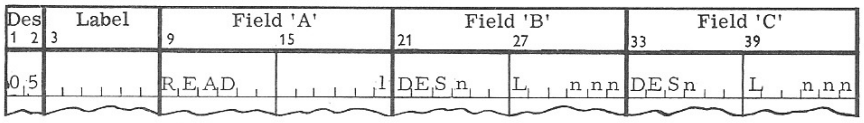
The basic format is



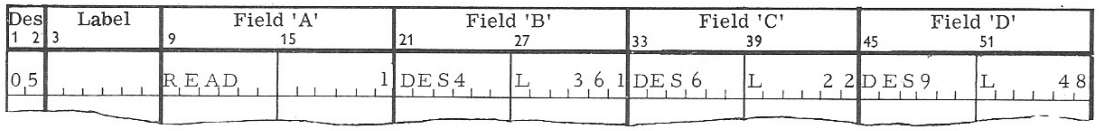
Note: This is a 'logical read', presenting cards to the program, and not necessarily a physical read. TAS 1, for instance, by its use of PPF-C, will seldom physically read cards singly.

In addition, keywords provide the facility to transfer control if so desired to various points in the program according to the designation of the card input.

The verb then takes its (optional but usual) extended format:



Example:



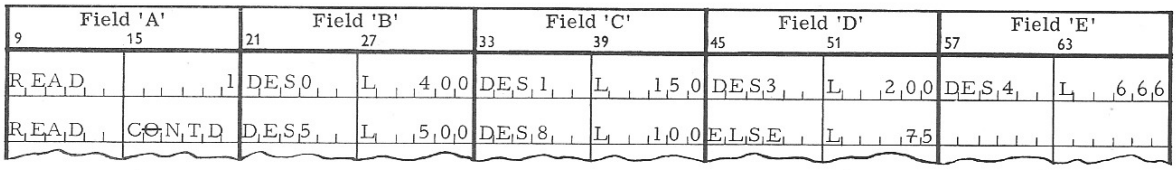
If the card input is type 4, control will be transferred to L..361, and so on.

These designations must be specified in *ascending* order. An unconditional transfer may be generated by setting the keyword 'ELSE..' in the field immediately following that containing the last 'DESn' conditional transfer, together with the label to which control is to be transferred should none of the individually specified designations occur.

Should the designation of a card input on any particular occasion not be specified, and no 'ELSE' condition be present, the program will merely continue to the next instruction in sequence.

Should more than four transfer instructions be required, the READ verb may be 'continued'. In field A of the next card is placed 'READ.. CONTD.', and the specification of transfers may then continue uninterrupted. Further continuations may be made if necessary.

Example:



Note: It is not necessary to specify in an extended Read order a conditional jump for all DES's described in input formats. Nor is it necessary in any format to distribute the actual designation itself, in order to transfer control according to its value.

In TAS 1 however (but not in TAS 2) it is necessary to describe each card type whose designation is to be used in this fashion. Thus one or more cards may be described as 'I80'

Modification

This verb may not be modified.

PUNCH 4.7.16

Only one card at a time may be punched out, its format type appearing in the second half of field B. The second half of field A must contain the number '1'.

The format is:

Des	Label	Field 'A'		Field 'B'	
1 2	3	9	15	21	27
0 5		PUNCH	1		CARDnn

where $01 \leqslant nn \leqslant 20$ in TAS 1 and $01 \leqslant nn \leqslant 30$ in TAS 2.

Modification

This verb may not be modified.

PRINT 4.7.17

This verb enables the programmer to output one or more lines on the printer, selecting the format types from those defined, and also specifying the spacing to be inserted between the lines. It is also possible to test 'paper trolley empty'. This is normally done before printing the first line of a form to find if sufficient paper remains in the machine to continue printing.

The verb is written in the first half of field A. In the second half is specified the number of lines to be printed. A maximum of 10 lines may be printed using a single instruction, up to four of which may be specified on an instruction card. If more than four lines are called at once, the instruction is continued on subsequent lines, by repeating the verb PRINT, and placing CONTD ('Continued') in the second half of field A.(Note - if the TAS 1 "B" Control Pack is to be used, then only one line at a time may be printed.)

Fields B, C, D and E contain, in the first half, the spacing to occur *before* the printing of the line whose format type is specified in the second half. This spacing is written as 'SPnn' in the first four characters of a field; up to 99 lines may be spaced before any line. It is possible to space 'zero' (SP00), and thus print two or more Formats on the same line.

The fifth and sixth positions of a field may contain 'PT' should the 'paper trolley empty' test be required to be carried out after spacing and before printing the line specified in the Field.

Format

Des 1 2	Label 3	Field 'A' 9 15		Field 'B' 21 27		Field 'C' 33 39		Field 'D' 45 51	
0,5		PRINT	n,n	SPnn	LINEnn	SPnn	LINEnn	etc.	

Example:

Field 'A' 9 15		Field 'B' 21 27		Field 'C' 33 39		Field 'D' 45 51		Field 'E' 57 63	
PRINT	7	SP02PT	LINE03	SP01	LINE04	SP01	LINE07	SP01	LINE01
PRINT	CONTD	SP04	LINE18	SP04	LINE20	SP02	LINE07		

Modification and Variable Spacing

Variable spacing before a line may be arranged by placing 'SP.VAR' in the first half of the relevant field, in place of 'SPnn'. This can be regarded as always zero, and is set up to the required value on each occasion by means of a MODIFY instruction. 'PT' cannot be tested in this case. This will be fully explained in Section 4.7.18.

With this exception, this verb cannot be modified.

Note:

(a) The spacing is (as it is in machine coding) the distance expressed in lines from one line of print to the next. It is not the number of clear lines left blank between them, i.e. 'SP01' will give printing on consecutive lines, 'SP04' printing on every *fourth* line, with three lines left clear, and so on.

(b) TAS has been so arranged that when the object program is running, it will ignore the very first spacing requirement it encounters, and print the line in the position to which the paper has been set by the operator. Should the program subsequently loop back to the same PRINT instruction, the space requirement will then be treated as normal and spacing will occur.

(c) The section on 'Runout' (4.7.2 above) should be noted carefully in connection with the PRINT verb.

MODIFY 4.7.18

This is the only verb by which program instructions may be altered at object program running time.

It may operate in two ways:

(a) Firstly it may operate on various other verbs to modify the *address* of the specified I.A.S. Field or Table (but not Working Store), by the *contents* of an I.A.S. Field or Working Store (but not of a Table). The format (which is fixed) is:

Des 1 2	Label 3	Field 'A' 9 15	Field 'B' 21 27	Field 'C' 33 39
0 5		MODIFY name A	IN Lnnn/z	WITH name B *

where z = A, B, C, D or E and refers to the Card Field of the Instruction 'Lnnn' which is the object of Modification.

Suppose we have the instruction

Des 1 2	Label 3	Field 'A' 9 15	Field 'B' 21 27	Field 'C' 33 39
0 5	L 999	MOVE 1	FROM X	TO A

then if A were located at I.A.S. 300, by modifying A in field C of the MOVE verb, by an I.A.S. Field which contained '5', we could cause the contents of X to be transferred not to I.A.S. 300 but to I.A.S. 305. Thus the MODIFY instruction would be:

Des 1 2	Label 3	Field 'A' 9 15	Field 'B' 21 27	Field 'C' 33 39
0 5		MODIFY A	IN L999/C	WITH ALTER

The Field named ALTER would hold '5'.

The action taken by the MODIFY verb is known as 'B-Line' modification, whereby the object instruction in its modified form is created and obeyed but not preserved, so that no demodification is necessary and the original skeleton object instruction remains constant throughout the program. Thus in the example we have taken above, 'A' would have the value 300 every time the instruction was arrived at; it would not be permanently altered to 305 by the action of our MODIFY verb. This means that if 'progressive modification' is desired, the modifier itself must be updated either before or after each occasion on which it is used. These modifiers are often simple counts.

Modification always takes place in terms of machine words, and not in terms of I.A.S. Fields or Table Items. Particular account of this must be taken when modifying references to Table Names.

(b) Secondly, it is possible to achieve variable spacing by modifying the number of lines to be spaced in a PRINT Instruction.

Instead of the 'SPnn..' which normally appears, the term 'SP.VAR' is inserted in the appropriate field of the PRINT Instruction. Then the MODIFY verb merely takes on the format:

Des 1 2	Label 3	Field 'A' 9 15	Field 'B' 21 27	Field 'C' 33 39
0 5		MODIFY SP VAR	IN Lnnn/z	WITH name A *

name A holding the number of lines it is desired to space on any particular occasion.

Various rules and restrictions apply to the MODIFY verb in both the above cases:

(a) Only data area *names*, or the special word 'SP.VAR', may be modified.

It is not possible to modify a card field section which *appears* as a *number*, e.g. the number of words to be moved or an Item number. To modify a Table Address, reference is made to the Table Name, which is usually installed in the MOVE instruction with no qualifying Item number.

Where an Item number appears in an instruction, it performs a permanent qualification of the address to which it refers. The modification of that address, which will refer directly to the name, performs a further temporary modification to the permanently qualified address. Suppose the object instruction above had read:

Des 1 2	Label 3	Field 'A' 9 15	Field 'B' 21 27	Field 'C' 33 39	Field 'D' 45 51
0,5	L 999	MOVE 1	FROM X	TO A	ITEM 7

then the form of the MODIFY verb would be

Des 1 2	Label 3	Field 'A' 9 15	Field 'B' 21 27	Field 'C' 33 39	Field 'D' 45 51
0,5		MODIFY A	ITEM 7	IN L999/C	WITH ALTER

and its effect would be to alter the MOVE so that the contents of X were transferred not to I.A.S. 306 but to I.A.S. 311. Exactly the same principles apply when Tables are involved.

(b) A MODIFY instruction must occur in the same block as its object and must precede it. Further, the object instruction may not be obeyed on any occasion without the instruction modifying it having been executed. It is impossible to modify an instruction on some occasions but not on others. Conditional modification may only be achieved by having a modifier which is conditionally zero. Finally, modification instructions must appear in the same physical order as the card fields containing the modified names. e.g. L013/B must be modified before L013/C. Similarly, if L...13 precedes L...10, the instruction modifying L013/C must precede one modifying L010/B.

(c) No field in any instruction may be the object of more than one modifier. There is a complete one-for-one relation between a MODIFY verb and its object.

(d) The label of the object instruction, and the reference to it in the MODIFY verb, must tally *exactly* as to punching. A punched zero and a space are not equivalent in this case.

PRES

PRES may not be referenced in a MODIFY instruction.

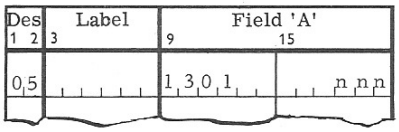
Modification

Data area names in this instruction may themselves be modified.

1301 4.7.19

This verb enables the programmer to include in his TAS program machine-coded instructions. These may be required to ensure that full efficiency is achieved, or to provide a link between TAS and large sections of machine coding held outside the I.A.S. area under TAS control (see Appendix F).

The format is completely fixed:



where nnn is the number of words of machine coding to be included.

This macro is punched on a TAS card which is immediately followed by normal 1301 relativised program cards containing the number of words of coding specified. These cards are treated as a separate section for the purposes of sequence numbering, being given numbers in column 26, commencing at 1.

The instructions are written entirely in machine code as far as Designations and Functions go, but the six columns allocated for Addresses and Relativisers may be used in any one of three ways:

(a) They may be used in 'absolute' form by referring to locations in the machine by their absolute addresses.

(b) They may be used in 'relative' machine code form; but in this case only relativiser 'B' (referring to locations within the current section of 1301 coding) may be used.

(c) Finally, normal TAS mnemonic references to I.A.S. Fields, Working Stores and labels, may be used. This means that one may combine the absolute efficiency of machine coding with the ease of reference of TAS mnemonics. Fields may not, however, be itemized nor may Tables be referenced, and any label referred to must be in the same TAS block, and precede the instruction referring to it.

Any two or all three of these methods may be used in the same section of 1301 Coding.

Every 1301 verb *must* be labelled. (Note: This refers to the TAS verb, *not* the machine code instructions themselves.) Control is transferred by means of a verb or verbs referencing this label, to the first word of the section of 1301 coding. No other entry point is permissible. Control may be transferred back either by the storing on entry of a link through which exit is made, or by a '4 00 L..nnn' instruction (care being taken to comply with the restrictions on this form set out above).

Positive constants in '1301' sections must not be designated.

Negative constants must be designated 'M', and must be in absolute denary form with a maximum length of eleven digits.

Zero constants must be designated 'P'.

Any word on a 1301 relativised program card which has been left completely blank will be ignored, and following words, if any, will be 'blocked up'.

Any number of sections of 1301 may be incorporated in a TAS Block, but they must be taken into account when calculating its size, which is still restricted to 200 machine words in all. Each section must be headed by a labelled 1301 Macro.

A 1301 section must be fed in at the end of the TAS block in which it appears. Should more than one section be present they must follow each other at the end of the block. The last section of 1301 will be headed by a '1301 ...nnn' macro as normal, and it is the label of this macro which will be referenced on the Block Heading Card. No 1301 relativised instruction may be labelled.

It is not possible to have an entire TAS block consisting of nothing but '1301' coding. If this is wanted, one 'dummy' TAS instruction, which will never be referenced or obeyed, must be placed at the head of the block. This 'dummy' instruction would normally be a 'STOP' (but note that in multi-block programs at least one 'GO TO' instruction will be required to transfer control to other blocks).

PRES

PRES may not be referenced in 1301 coding.

Modification

A 1301 macro may not be the object of the MODIFY verb, nor may any 1301 instruction.

Example

I.C.T COMPUTERS

1300 SERIES PROGRAM SHEET

JOB:- Label 827

BLOCK No.

SHEET No. 1 /

PROGRAMMER:-

/ /

C	I	D	F	A	R	NARRATIVE
1	0		37	17	B	
			42	800		
	1		60	810		
		4	01	L 4	32	
	2		37	WS	2	
			42	801		
2	3		21	0		
			69	18	B	
	4		42	WS	1	
		4	03	15	B	
	5		45	ANSW	ER	
			01	17	B	
3	6		37	WAGE		
			42	21	B	
	7	4	00	L	24	
	8	P				
4	9	M				
				47		

FORM No.1/1666

Printed in Great Britain by International Computers and Tabulators Limited, 149 Park Lane, London, W.1.

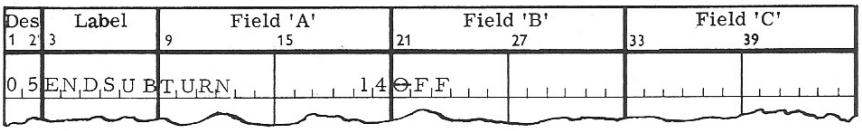
Subroutines and the verb OBEY 4.7.20

The general concept of Subroutines has been discussed in 4.6 above. Subroutines in TAS are distinguished by the fact that each is headed by a Subroutine Identification Card, bearing the word 'SUB...' in the Label Column, and in the second half of Card Field A a unique identification number. Like Instruction labels, these numbers must be unique throughout the program. They may, again like labels, be allocated randomly, except that for each different type of Subroutine in TAS a different range of numbers must be used.

Three forms of Subroutine exist in TAS: the Block Subroutine, and the Global Subroutine, both written in TAS; and the Library Subroutine, which is written in machine code, and is generally, but not necessarily, a standard I.C.T. Subroutine available from the Subroutine Library.

Endsub

This is a special label given to the last instruction of each subroutine presented to the compiler. It is written in the label column of the last instruction:



It has three functions:

(a) It informs the compiler that the subroutine has come to an end, just as the label of the last instruction of a block informs the compiler of the end of that block.

(b) Again like the label of the last instruction of a block, it does, in fact, label the instruction and may be referenced in other instructions within the subroutine as a normal label. Note that since there will be one 'ENDSUB' in every subroutine, there may well be more than one 'ENDSUB' in a program. For this reason it is not permitted to reference such a label except from within the particular subroutine of which it is the last instruction.

(c) In a Block subroutine only, it causes the compiler to generate a jump to the subroutine 'Link'. This jump is stored immediately following the instruction of which 'ENDSUB' is the label. In a *Global* subroutine, no automatic jump is generated and there are also certain restrictions on the 'ENDSUB' instruction itself; these are stated below.

Block Subroutines

If the use of and references to a subroutine fall wholly within a single block of program, then the TAS-coded Block Subroutine may be used. It forms part of the block from which it is referenced and must therefore be taken into account when calculating the size of that block.

Up to 20 block subroutines may be included in one block, provided the block maximum size is not exceeded. They must always be placed at the head of the block in which they appear. All linkage is taken care of by the compiler, which automatically inserts a transfer of control back to the stored link after the 'ENDSUB' instruction.

A Block Subroutine must be allocated an identification number in the range 150-199, the order being immaterial.

Global Subroutines

Sometimes it is not possible to contain all references to a TAS-coded Subroutine within a single block, and in this case a Global subroutine must be used. This is treated as a separate block of program, the Block Heading Card of which contains the word 'ENDSUB', instead of a Label Number, in the second half of field A.

Thereafter it is written in the same form as a Block Subroutine, headed by a 'SUB' card, and the last instruction labelled 'ENDSUB'. It must be given an identification number in the range 200 to 299.

Linkage is taken care of by the compiler, except that the automatic transfer of control back to the link is *not* inserted after the 'ENDSUB'. The instruction 'GO TO LINK' must therefore be inserted by the programmer after the last (logical) instruction.

Important restrictions on Global Subroutines are:

(a) It is not possible to transfer control to a point in a Global Subroutine other than the beginning, as no LINK will be set, and any set previously will not have been preserved.

(b) Various verbs may not be used. In TAS 2, it is not possible for a Global Subroutine itself to contain the verb OBEY. In TAS 1, the use of the verbs OBEY, READ, PUNCH and PRINT is forbidden.

(c) It is not possible to enter or exit from a Global Subroutine with relevant information held in PRES. The contents of PRES are altered by the control transfer routines.

(d) The 'ENDSUB' of a Global Subroutine must be either a 'GO TO', or a 1301 macro specifying one word of machine coding. In the latter case, the word itself must be a constant of zero, and must not be used by the program.

(e) When compiling, Global Subroutines must come after all other blocks.

Library Subroutines

Usually machine-coded subroutines may be handled as sections of '1301' coding, but sometimes it is unavoidable that a separate block is formed. In this case it must be treated as a 'Library Subroutine'.

This subroutine will not itself be fed into the compiler with the source program. Instead, it need merely be referenced, like a Block or Global Subroutine, by the OBEY verb. It must be given an identification number in the range 300 to 399.

The subroutine will then be allocated a drum channel, and references to it will cause transfers of control to word 1 of that channel. The compiler will punch out a card containing the identification number given by the programmer, together with a 'B' word containing the drum start address allocated.

The programmer will then insert the Subroutine into his compiled object program by placing his (prepunched) machine-coded pack immediately after this card.

It will also be necessary to insert, immediately before this card, a card setting up any relativisers required by the Subroutine. The first of these may well be RRN 1, Temporary Storage. This must be allocated an area between I.A.S. 0 and I.A.S. 199, i.e. within the block. It must not, of course, be set so as to overwrite any part of the Subroutine, which will itself be stored here, commencing at word 1.

From this it will be clear that it is possible to use this form only for those routines which, with their associated temporary storage, do not exceed 200 words of I.A.S. For those which do, the techniques suggested by Appendix F will need to be employed.

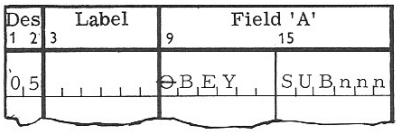
A more complicated problem may be set by Input and Output parameters. When there is only one parameter of either type, or when the particular pieces of information with which the routine is required to deal lie in the I.A.S. both adjacent to each other and in the order required by the routine, then no difficulty arises. The user merely sets the relativisers to the appropriate absolute addresses allocated by the compiler to the various Fields, which may be obtained from the Field Names List print-out.

Unfortunately it is not always convenient or even possible to arrange the I.A.S. in this fashion, and, if this is so, the technique described below as the extended form of the verb OBEY must be adopted.

The same restrictions on the contents of PRES during entry and exit apply here as for Global Subroutines. It may therefore be necessary to make small alterations to certain routines to overcome this problem.

OBEY

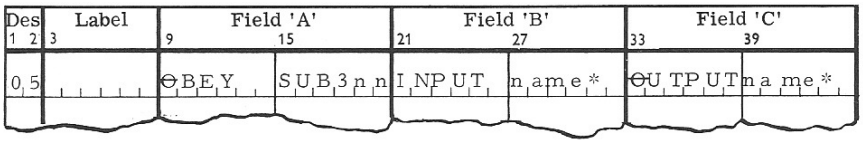
Control is transferred to a Subroutine by means of this verb, whose basic format is simply:



where nnn is the identification number of the Subroutine which it is desired to enter.

This will cause control to be transferred to the beginning of the Subroutine when a link will be stored. Eventually this link will be entered, and control will be returned to the main program at the instruction immediately following the verb OBEY.

Notice that the same verb is used in the same way for Block and Global Subroutines. For Library Subroutines only, however, an extended Format must be used. All Library Subroutines require Input and Output parameters which must be set under relativisers as detailed in the specification sheets. To cater for the cases mentioned above, where the I.A.S. cannot be so arranged as to allow the 'direct' setting of these relativisers, TAS requires the user, when transferring control to a Library Subroutine, to specify these parameters. The verb therefore takes the extended format:



The relevant names are written, in the order required by the subroutine, after each key word. Should it not be possible to detail all the parameters in this fashion on one line, then the instruction may be extended over two or more. The verb is repeated in the first half of card field A, in the second half of which is placed the word CONTD. ('continued'). The specification of parameters then continues as if there had been no interruption.

The effect of these specifications will be to cause TAS to arrange a program which will, before entering the Subroutine, move the contents of the fields specified after 'INPUT' into I.A.S. locations 390 to 399 (TAS 2) or 395 to 399 (TAS 1) in the order specified. After exiting from the routine, another program will move the contents of the words 395 to 399 to the locations specified after 'OUTPUT'.

Notice that if this facility is used:

(a) The programmer must set his Input parameter relativiser to I.A.S. 390 (TAS 2) or 395 (TAS 1) and his Output to 395.

(b) There is a limit of 10 (TAS 2) or 5 (TAS 1) Input parameters, but the last five of both are overwritten by the Output. There is a limit of 5 Output parameters.

(c) The Keyword 'OUTPUT' and the various names may appear in either half of any card field but 'A'.

(d) Any name may be Itemized, but neither the second half of field E nor the first half of field B may contain the word ITEM. That is, an Itemized Field must be wholly described on one card. In order to achieve this, half, or all, of field E may be left blank.

(e) Table names may not be specified.

If this facility is not used (because both groups of parameters have been set up 'directly') then the keywords INPUT and OUTPUT must still appear even though not followed by any field name:

Des	Label	Field 'A'		Field 'B'		Field
1 2	3	9	15	21	27	33
0,5		OBEY	SUB n,n,n	INPUT	OUTPUT	

Because the OBEY verb transfers control to the *first* word of a subroutine and only to the first word, it may be necessary on occasion to alter an I.C.T. standard subroutine with more than one entry point, or a single entry point which is not the first word of the routine, to conform to TAS requirements. Standard subroutines using Indicator 19 will also need to be altered. These are usually simple matters.

Certain Standard Subroutines zeroize output parameters on entry, and others use input parameters after storing one or more results. Care must be taken to program round this if necessary.

Restriction

The verb OBEY may not be used in a Global Subroutine.

Modification

No name in an OBEY instruction may be modified.

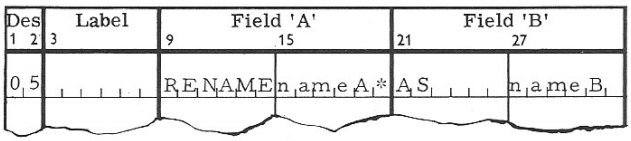
Choice of Subroutine Type

When deciding what type of subroutine to use, the programmer should bear in mind the fact that 'OBEY'ing a Block Subroutine (or its machine code equivalent, a section of '1301') involves no drum transfer as such. On the other hand, 'OBEY'ing Global or Library Subroutines involves three, one to store the state of the block from which the routine is entered, one to bring the routine to the I.A.S., and one more to restore the original block. It is obvious that if it is possible to avoid these drum transfers, the result will be a considerably improved object program running speed. It may well be advantageous to repeat one subroutine several times in different blocks as a Block Subroutine or Section of 1301, rather than holding it once separately.

RENAME 4.7.21

This is a 'pseudo-verb' which causes no action during the object program. It is designed to enable the programmer entering a new section of program to re-use I.A.S. Fields which have become redundant, without being forced to refer to them by names which have lost their significance.

The instruction has the fixed format



where name A is the obsolete name, and name B the name by which it is desired to refer to the Field in the future. Name B may not be Itemized.

This instruction is fed in as a normal instruction card of the procedure section; the compiler will treat all references on subsequent cards to name 'B', as being to the Field formerly referred to as name 'A'. Note: Should name 'A' be referred to again it will not be recognised and will be allocated a new address.

There is no limit to the number of times a Field may be renamed, and no reason why it should not, at some stage, revert to one of its previously discarded 'aliases'.

On the Field Names Directory (see 6.2.2 below) only the name in use at the end of compilation will be shown.

It is possible to rename only I.A.S. Fields. In addition, this facility may not be used to 'RENAME' a field originally defined in a Re-defining or Sharing Record (see 5.2.2 and 5.2.3 below).

Modification

As 'RENAME' is not an instruction obeyed at object program running time, it is obvious that no name mentioned in it may be modified.

Chapter 5

MAGNETIC TAPE

Whilst handling Magnetic Tape is in no way difficult, the macros are at a high level, and for this reason the user would be best advised not to attempt to read this chapter before he has a full working knowledge of all other aspects of TAS.

A Tape File consists of Records arranged in groups, or 'blocks' on the tape according to a standard format (see Appendix M). Any number of different types of Records, of the same or different lengths, arranged in any order, may go to make up one File.

TAS 2 provides the user with a powerful system for handling tapes. The TAS programmer does not have to concern himself at all with the problem of the 'physical' handling of tape blocks; he merely regards his Files as strings of Records. As far as he is concerned, the tape verbs perform 'logical' reads and writes just as the other input/output macros perform 'logical' reads, prints, and punches.

At the same time he retains control over questions of space and speed, and may select batched or unbatched File and/or Record areas, and a greater or lesser degree of simultaneity in reading and writing his tapes. However, it should be noted that while read/write overlap may be achieved, time-sharing between tape transfers and all other processing has, because of the dangers of drum and tape overlap, been locked out.

This chapter sets out what the user must do to enable the compiler to create the tape handling system, the system itself, and also certain aspects of setting up and running a TAS Tape Program.

TAPE FILE DESCRIPTIONS 5.1

The user must describe each of his Tape Files on the Sheets provided (see Appendix H). Up to 8 different Files may be described.

For any File, the user must describe four things

(a) Whether the File is an 'INPUT' or 'OUTPUT' Tape. No one File may be described as both.

(b) The 'FILE NAME', by which it will be referenced in the program. This must obey the rules set out in Appendix A, but need not be the name on the Beginning of File Label and Job Set-up Identity Cards.

(c) The 'LENGTH' in words of the largest physical block on that File. This may vary between 6 and 449, and includes records and the 'Highest Key' word, but *excludes* the end-of-block marker. This enables the compiler to allocate I.A.S. storage for the File buffer area. When so doing, it automatically allocates one extra word for the end-of-block marker, and one (for an input tape) or two (for an output tape) words for TAS keys (these are for use by the control, and are not written to tape). The user need concern himself with none of these.

(d) The tape DECK ADDRESS of the File, any single digit in the range 1 to 8. Deck Addresses, like labels, may be allocated randomly, but only one File may be allocated to any one Deck Address.

In addition, for each *Output* File the user must tell the compiler where the 'key word' is to be found in each record. This is done by writing 'KEY.nn', thus instructing the tape control program to update the word holding the 'highest key' by the 'nn'th word of each record as it is stored in the output buffer area. If 'nn' is set at '00', this will not take place, and the word holding the highest key will be zeroized.

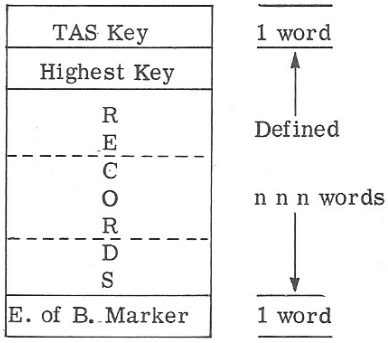
There are four methods of describing Files, one for each system available to the user. Any or all may be used for different Files in the same program. They are described in detail below.

Independent Reading (Input File) 5.1.1

This format is used when the user does not wish the control program to time-share the physical reading of the File described with any physical tape writing.

Des 1 2	3	Input or 9 Output	File 15 Name	21	Block 27 Length	Deck 33 Address	Position of 39 Keyword
0 6	F I L E	I N P U T	n a m e	L E N G T H	n n n	D E C K x	

Allocated Buffer Area



Example:

Des 1 2	3	Input or 9 Output	File 15 Name	21	Block 27 Length	Deck 33 Address	Position of 39 Keyword
0 6	F I L E	I N P U T	M A I N F L	L E N G T H	2 0 0	D E C K 2	

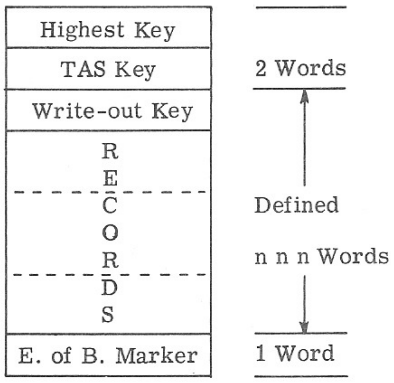
This will cause the compiler to allocate a 202 word buffer area for an Input File called 'MAINFL held on Deck Address 2. As described, this File *cannot* be used to achieve simultaneous Read/Write.

Independent Writing (Output File) 5.1.2

This format is used when the user does not wish the control program to time-share the physical writing of the File described with any physical tape reading.

Des 1 2	3	Input or 9 Output	File 15 Name	21	Block 27 Length	Deck 33 Address	Position of 39 Keyword
0 6	F I L E	⊖U T P U T	n a m e	L E N G T H	n n n	D E C K x	K E Y n n

Allocated Buffer Area



Example:

Des 1 2	3	Input or 9 Output	File 15 Name	21	Block 27 Length	Deck 33 Address	Position of 39 Keyword
0 6	F I L E	⊖U T P U T	U P D A T E	L E N G T H	1 7 6	D E C K 1	K E Y 1 0

This will cause the compiler to allocate an I.A.S. buffer area of 179 words for an output File called 'UPDATE' on Deck Address 1.

The word holding the highest key will be made equal to the 10th word of each Record as it is stored in the buffer area. As described, this File *cannot* be used to achieve simultaneous Read/Write.

Simultaneous Read/Write 5.1.3

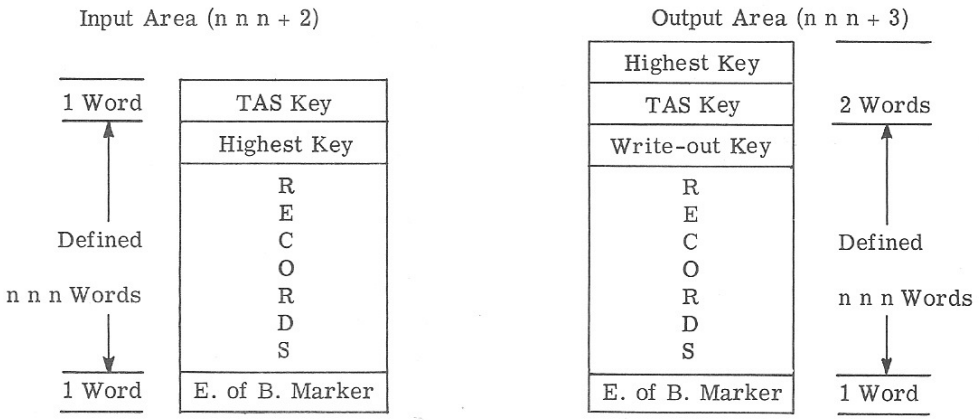
An Input and an Output File can be made to achieve simultaneous Read/Write by the use of 'SIM'. TAS allows the user to choose between two alternatives offering different degrees of simultaneous Read/Write, basing his decision on the amount of I.A.S. available. The two Files on which simultaneity is desired must be described on consecutive cards, the input File first.

Any one input File may be described as 'SIM' with only *one* output File, and vice versa. The same File may not be described in more than one 'SIM' pair.

Simultaneous Read/Write (Single Output Area) 5.1.4

Des 1 2	3	Input or 9 Output	File 15 Name	21	Block 27 Length	Deck 33 Address	Position of 39 Keyword	45 Sim.	Associated 51 File	Single or 57 Double 62
0 6	FILE	INPUT	name A	LENGTH	nnn	DECK x		SIM	name B	
0 6	FILE	~~O~~UTPUT	name B	LENGTH	nnn	DECK x	KEY nn	SIM	name A	SINGLE

Allocated Buffer Areas



Example:

Des 1 2	3	Input or 9 Output	File 15 Name	21	Block 27 Length	Deck 33 Address	Position of 39 Keyword	45 Sim.	Associated 51 File	Single or 57 Double 62
0 6	FILE	INPUT	M~~O~~VEIN	LENGTH	200	DECK 3		SIM	M~~O~~VE~~O~~T	
0 6	FILE	~~O~~UTPUT	M~~O~~VE~~O~~T	LENGTH	100	DECK 4	KEY 05	SIM	M~~O~~V EIN	SINGLE

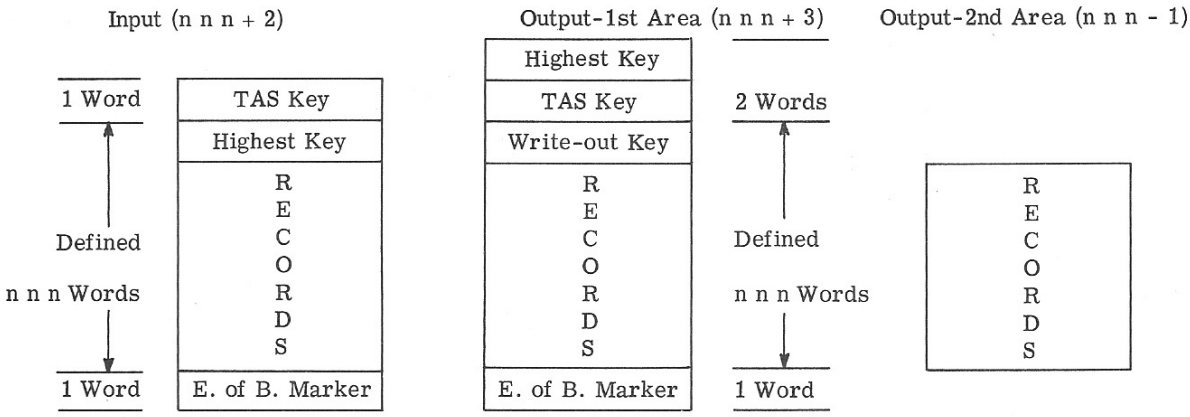
This will cause the compiler to allocate 202 words of buffer storage for the input File called 'MOVEIN' on Deck Address 3, and a single buffer area of 103 words for the output File called 'MOVEOT' on Deck Address 4. The 'highest key' word in the output buffer area will be made equal to the 5th word of each record as it is stored in the buffer.

This method of describing Files will achieve simultaneous Read/Write only if the control finds the input buffer area empty at the time when it is necessary to clear the output buffer. Hence, to achieve full simultaneity, the output buffer should be one record ahead of the input buffer.

Simultaneous Read/Write (Double Output Area) 5.1.5

Des 1 2	3	Input or 9 Output	File 15 Name	21	Block 27 Length	Deck 33 Address	Position of 39 Keyword	45 Sim.	Associated 51 File	Single or 57 Double 62
0 6	FILE	INPUT	name A	LENGTH	nnn	DECK x		SIM	name B	
0 6	FILE	OUTPUT	name B	LENGTH	nnn	DECK x	KEY nn	SIM	name A	DOUBLE

Allocated Buffer Areas



Example:

Des 1 2	3	Input or 9 Output	File 15 Name	21	Block 27 Length	Deck 33 Address	Position of 39 Keyword	45 Sim.	Associated 51 File	Single or 57 Double 62
0 6	FILE	INPUT	UPDATE	LENGTH	100	DECK 5		SIM	MOVEOT	
0 6	FILE	OUTPUT	MOVEOT	LENGTH	200	DECK 6	KEY 00	SIM	UPDATE	DOUBLE

This will cause the compiler to allocate 102 words of buffer storage for the input File called 'UPDATE' on Deck Address 5, and *two* buffer areas, totalling 402 words, for the output File called 'MOVEOT' on Deck Address 6.

Notice that not only are the 3 extra words not allocated to the second area, but neither is the 'highest key' word. Notice also that, since we have described the key word (KEY.nn) as zero, the 'highest key' word in the output buffer area will be zeroized before writing to tape.

This is the description method to use for the best tape handling speeds. Output Records are stacked in the first buffer area until it is full, when they are stacked in the second. When the input buffer area is empty and the first output one is full, simultaneous Read/Write takes place. The records in the second output buffer area are then moved into the first, and processing continues. Only when both output buffers fill before the input buffer empties will writing take place without reading.

File Buffer Areas: Storage Allocation 5.1.6

File 'buffers' will be allocated I.A.S. storage in the same order as the File descriptions, starting immediately after the standard control program storage. As there is no environment division in TAS 2, no checks can be made to ensure that the requested storage fits into the available I.A.S.

TAPE RECORD FORMATS 5.2

Tape Records are described on Format sheets, the general principle of description being the same as that for other Input or Output formats. In the TYPE column is written 'RECz..', where 'z' is the code letter (in the range A to Z) of the format. In this way may be described up to 26 different Record formats. No distinction of any kind is made between input and output Records, and no connection exists between particular Records and Files. Any one Record may be read from or written to any, or all, Files. Conversely, any number of different Records may be read from, or written to, any one File.

Starting on the same line in the 'KEY' column is written the description key of the first Field of the Record, and again on the same line, the name chosen for that field. This name may not be that of a Table or Working Store and may not be Itemized. Subsequent keys and names are written on subsequent consecutive lines. The Record must be described completely, starting at the first word and specifying the Fields in order.

At object program time (on request) the control program will merely present to the user, or stack away as appropriate, the next Record of a File to, or from, the Record area named. No rearrangement or reorganization of the actual words or digits of the data will take place, and, for this reason, the KEY contains only one piece of information. In positions 5 and 6 is written the number of data words occupied by the Field, the first four positions being left blank. Any one Field may be up to 99 words long.

From this it will be clear that automatic packing and unpacking of data does not take place, all Fields being presumed to be one or more complete words in length.

As with other formats,'RECz..' must be repeated at the head of each card needed to describe the Record, and each Record format must commence a new card.

There are three methods of describing records, and these are shown in 5.2.1, 5.2.2 and 5.2.3.

A Record having its own Record Area 5.2.1

DES 1 2	TYPE 3 8	KEY	FIELD NAME	CARD No. COLS 69-74	
07	RECA	/ 1	PLANT		
		/ 1	STORE		
		/ 1	PARTNO		
		/ 3	DESCR		
		/ 1	QTY		
07	RECA	/ 1	PRICE		
		/ 1	QTYCOD		
		/			

This will cause the compiler to allocate 9 words of I.A.S. storage for a Record referenced as 'RECA', split up into Fields referenced by the Field names described in the column headed 'FIELD NAME'. These Fields are used and referred to as normal I.A.S. Fields.

A Record re-defining another Record Area 5.2.2

In order to save storage, it is possible to make all, or a number of Records share the same Record area. In this case, the 'original' Record area must be equal to, or greater than, any other re-defining it.

DES 1 2	TYPE 3 8	KEY	FIELD NAME	CARD No. COLS 69-74	
07	RECN/A	/ 1	PLANT		
		/ 1	STORE		
		/ 1	PARTNO		
		/ 1	USAGE1		
		/ 1	USAGE2		
07	RECN/A	/ 1	USAGE3		
		/ 1	QTY		

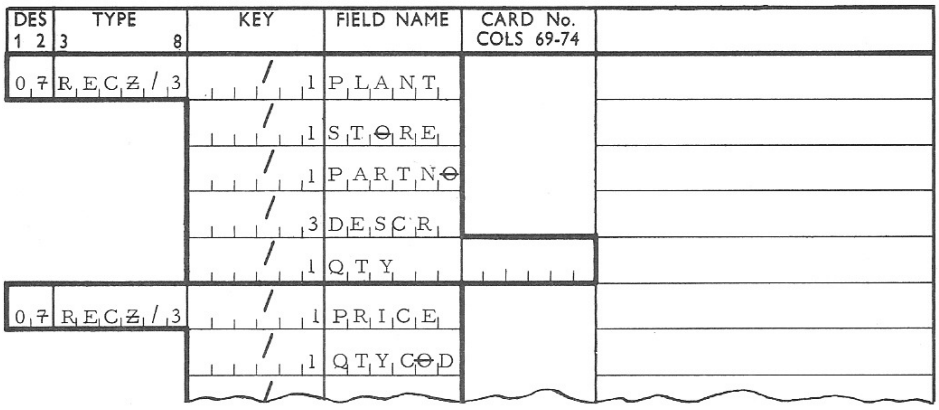
This description will cause the compiler to recognise the Fields described, but not to allocate any new storage for them.

In this way is provided an automatic 'Rename' facility for complete or part Records, with the additional benefit that the original name or names are not lost; both old and new names may be referenced throughout the program.

Note: Although it is possible to 'WRITE' any Record to a File, it is not possible to 'READ' a Re-defining Record. The procedure must always be to read the original Record, and then refer to the Fields in the re-defining one.

A Record 'Sharing' a File Buffer Area 5.2.3

It is also possible to make a Record 'share' the buffer area of the input File from which it is read. This File may not be 'SIM'. This is done by writing in position 6 of the TYPE column the Deck Address of the relevant File.



This description will cause the compiler to recognise the Fields described and listed, but not to allocate any new storage for them. The Record area for 'RECZ' will be in the buffer allocated to the File on Deck Address 3, starting at the first word of the first Record in that File buffer.

Only one Record may share any one File area. Other Records can be made to use this area only by using the re-defining facility: e.g. to cause 'RECY' also to use the File area allocated to Deck Address 3, the user must specify 'RECY/Z'. 'RECY/3' would not be allowed.

No Record may share the File area of a 'SIM' File.

General Considerations 5.2.4

From the Record descriptions in the last three examples, it will be seen that the designatory letter describing a field as Denary, Sterling etc. is not present as with other Input/Output formats. This letter is not necessary here, since any Field mentioned that is associated with other input/output devices will be fully described (and appropriate distribution keys generated) in those formats.

Notice that, as with any other alphabetic constant, alphabetic information from tape which is to be output to either the printer or the punch must be in the form 6Z, 6N.

The control program does not 'distribute' a tape Record as it does, for example, a punched card. It presents it, into the Record area, as a block of words in I.A.S. For this reason Fields in different Records may not be given identical names, except in the case of Re-defining Records with the Fields concerned occurring in the same position in both Records.

Apart from this, the Fields may be freely referenced in other input/output formats, and throughout the procedure, as normal.

The user must ensure that the first word of any Record contains the 'number-of-words' parameter (see Appendix M).

TAPE VERBS 5.3

As has been stated, the user does not have to concern himself with the problems of physical handling of tape blocks; he merely regards his Files as strings of Records, and the 'READ' and 'WRITE' verbs perform, as far as he is concerned, logical presentations and storings. This still applies to multi-reel files, as 'end of reel' conditions are dealt with by TAS, and do not therefore require user intervention.

End of File, on the other hand, does. Because any macro which may cause a physical read of tape may result in this condition occurring, the user must specify with any such macro an 'END condition label': 'AT.END.L..nnn'. This is the label to which control will be transferred should the condition arise. It must be in the same block as the macro referring to it. When the program reaches label nnn, the Deck Address of the File closed will be in digit 12 of PRES.

'READ RECord' and 'READ IN' are not the only macros which must have this. Any WRITE verb referring to a File 'SIM' with another may find the input buffer area empty and so perform a physical read of tape. Thus the end label of an input file may be reached on a WRITE macro, and for this reason any WRITE of a 'SIM' file must cater for the condition.

Modifications

No name in a tape handling macro may be modified.

READ RECord 5.3.1

This macro has the effect of presenting the next Record of the File named in field B to the Record area referenced. The File *must* be an input File.

The format is

Des	Label	Field 'A'		Field 'B'		Field 'C'	
1 2	3	9	15	21	27	33	39
0 5		R E A D	R E C z	F R O M	n a m e	A T E N D L	n n n

Example:

Des	Label	Field 'A'		Field 'B'		Field 'C'	
1 2	3	9	15	21	27	33	39
0 5		R E A D	R E C A	F R O M	M A I N F L	A T E N D L	1 0 9

This will present the next Record from the File named 'MAINFL' to the Record area 'RECA'. Should the 'end' condition occur, control will be transferred to label 109.

Note that, even if the different Records on a File do not all re-define the same Record area, it is often not possible to know in advance which particular Record is next in sequence. In this case, the procedure must always be to 'READ' the longest, and then discover by program which has been presented. There is no facility akin to the 'DES' test in card reading, for discovering which Record is in hand and distributing and transferring control accordingly.

If a Record has been defined as sharing an input File buffer, then that Record may be read from that File only, and from no other.

READ IN 5.3.2

This macro is used mainly, but not exclusively, for searching a reference File. It physically reads from tape to the File buffer area the next *group* of Records on the File, no matter whether the buffer area is empty or not. Thus, it enables the user to ignore the unwanted remainder of a group rather than clear it from the buffer Record by Record. A fuller explanation of its use is given in 5.4.2 below.

The format is

Des	Label	Field 'A'		Field 'B'		Field 'C'	
1 2	3	9	15	21	27	33	39
0,5		READ	IN	FROM	name	AT END	L nnn

The File must be an Input File.

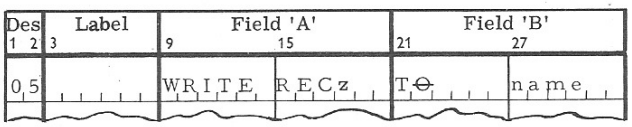
Example:

Des	Label	Field 'A'		Field 'B'		Field 'C'	
1 2	3	9	15	21	27	33	39
0,5		READ	IN	FROM	LIST	AT END	L 720

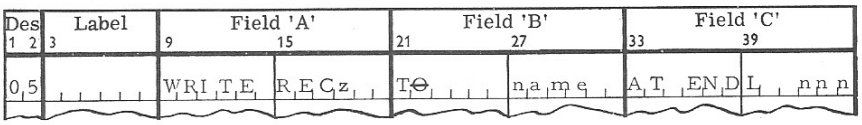
This will present the next group of Records from the Input File 'LIST'. Should the 'end' condition occur, control will be transferred to label 720.

WRITE RECord 5.3.3

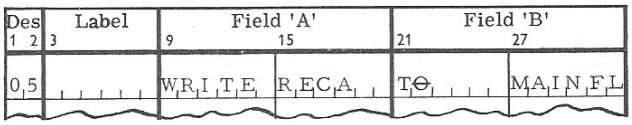
This macro has the effect of writing the Record area referenced to the File named in field B. This File *must* be an output File. The format is:



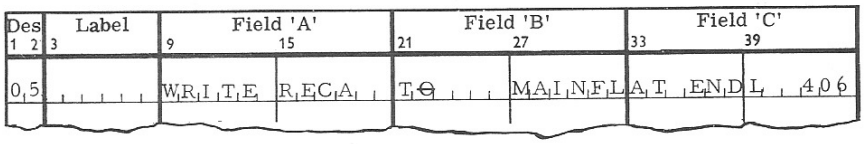
If the File is 'SIM' the format must be extended to cater for the END condition in the associated input File:



Example:



This will write 'RECA' to the File named 'MAINFL' which is *not* 'SIM' with an input File.



This will have the same effect as the previous example with regard to 'RECA'. However, 'MAINFL' is in this case 'SIM' with an input File, and therefore the extension to the format is necessary. Should an 'end' condition occur on the input File, control will be transferred to label 406.

WRITE AWAY 5.3.4

This macro is used to clear an output File buffer area, in order to keep different sections of a File separate on a tape, or to complete a File before writing an 'end' label to it.

It physically writes to tape all those Records remaining in the buffer area of the File named in field B. The File *must* be an output File.

Des 1 2	Label 3	Field 'A' 9 15		Field 'B' 21 27	
05		WRITE	AWAY	TO	name

If the file is 'SIM' with an input file, the format must be extended to:

Des 1 2	Label 3	Field 'A' 9 15		Field 'B' 21 27		Field 'C' 33 39	
05		WRITE	AWAY	TO	name	AT END	L nnn

Examples:

Des 1 2	Label 3	Field 'A' 9 15		Field 'B' 21 27	
05		WRITE	AWAY	TO	MOVEOT

This will write to tape all those records in the buffer area of the File 'MOVEOT'.

Des 1 2	Label 3	Field 'A' 9 15		Field 'B' 21 27		Field 'C' 33 39	
05		WRITE	AWAY	TO	MOVEOT	AT END	L 127

'MOVEOT' is now 'SIM' with an input File. The macro will have the same effect as the above with regard to 'MOVEOT', but should an 'end' condition occur on the input File, control will be transferred to label 127.

WRITE file TO file 5.3.5

This macro is used mainly, but not exclusively, for scanning a File which is being updated (see below, 5.4.2). It has the effect of writing all the Records not yet presented from the buffer of the input File named in field A to the output File named in field B. When this is completed, the input File buffer area is refilled. The format is

Des 1 2	Label 3	Field 'A' 9 15		Field 'B' 21 27		Field 'C' 33 39	
05		WRITE	name	TO	name	AT END	L nnn

The File named in field A *must* be an input File, and that in field B an output File.

The Files must be either independent or 'SIM' with *each other*. Neither File can be 'SIM' with a third.

Example:

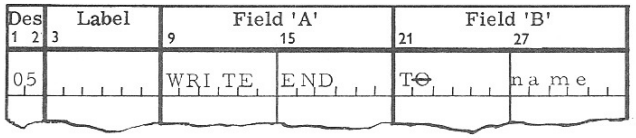
Des 1 2	Label 3	Field 'A' 9 15		Field 'B' 21 27		Field 'C' 33 39	
05		WRITE	MOVEIN	TO	MOVEOT	AT END	L 163

This will write all the Records not yet presented, from the buffer area of the input File 'MOVEIN' to the output File 'MOVEOT', and refill the 'MOVEIN' buffer.

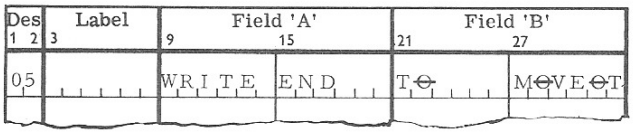
Should an 'end' condition occur on 'MOVEIN', control will be transferred to label 163.

WRITE END 5.3.6

This macro writes an 'end' label to, and closes, the File specified, which *must* be an output File. The format is



Example:



This will write an 'end' label to and close down the output File 'MOVEOT'.

It should be noted that a 'WRITE AWAY' should be given immediately before a 'WRITE END', in order to clear the buffer area before closing the File.

Once a 'WRITE END' order has been given to a File, that File must not be written to again, nor may it be read again without first being 'renewed'. To do so will be to bring the program to an inescapable 110920 stop.

WRITE DUMP 5.3.7

This macro dumps the current state of the computer onto any output File. The amount of drum to be dumped is at the user's discretion, but must cover the full object program, and any data areas lying beyond it on the drum. The format is

Field 'A'		Field 'B'		Field 'C'		Field 'D'		Field 'E'	
9	15	21	27	33	39	45	51	57	63
WRITE	DUMP	TO	DECK x	BLOCK	nnn	IAS	iii	DRUM	dddd

where x is the Deck Address of the selected File.

nnn is the maximum block size in words (excluding the end of block marker) during dumping. This may not exceed 210 words for $\frac{1}{4}$" tape programs, and in any case may not exceed the data block size of the Tape as defined in the File Description.

iii is the size of the I.A.S. in decades.

dddd is the number of decades of the drum, starting from word 0, to be dumped.

Example:

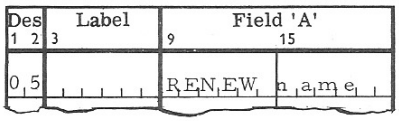
Field 'A'		Field 'B'		Field 'C'		Field 'D'		Field 'E'	
9	15	21	27	33	39	45	51	57	63
WRITE	DUMP	TO	DECK 3	BLOCK	150	IAS	1200	DRUM	10000

This will dump onto the File at Deck Address 3 (which is defined as having blocks of 150 words), 1200 words of I.A.S. and the first 10,000 words of the drum.

Note: A special drum Table must be defined by the user for use with this macro. (See Appendix Q.)

RENEW 5.3.8

This macro enables the user to re-commence the processing of an Input File from the beginning, at any stage of the program. The format is:



The File specified will be rewound, and re-opened as an Input File. (If the reel of the File attached is found to be other than the first, a stop will occur, and the operator may then re-attach the first reel.)

On exit, the Beginning of File Label will be left in I.A.S. 390 to 395.

'Renew'ing an Output File

The problem is that it is not possible in a TAS program to 'Read' from a File described as 'Output'. Therefore, in order to Read a File after having Written it, the following procedure must be followed:

(a) *Two* Files of different names, one Input, one Output, must be defined on the same deck. This is the only time two Files can be defined on the same deck, and the Input File must be the first to be presented to the compiler. Job Set-up is not performed on the *Input* File, which is not originally present.

(b) After 'Writing End' to the Output File, the order 'Renew' is given to the Input File and the program may then proceed to 'Read' from this File.

End *must* be written to an Output File before it is Renewed in this way. To fail to do so will be to cause the object program to break down. A multi-reel output File may be Renewed only on a deck on which was originally held an input File of identical Beginning of File Label. Job Set-up must have been performed on this File.

Note: Once a File has been 'end'ed, either by Reading or by Writing End, it is locked out, and may not be addressed again without first being Renewed. To attempt to do so will be to bring the program to an inescapable error stop.

PROGRAMMING CONSIDERATIONS 5.4

Highest Key Word 5.4.1

No word in a File buffer area as such may be addressed directly. Information off Files must be 'Read' into a Record area, and addressed from there.

The only exception to this rule is the 'Highest Key' of a Record Group (Tape Block). This may be referenced in TAS instructions by the name of the File itself, provided that it is not altered or updated in any way.

Example:

Des 1 2	Label 3	Field 'A' 9 15		Field 'B' 21 27		Field 33
0 5		LOAD	MOVEIN	etc		
0 5		COMP	MOVEIN	WITH	X	etc

Modification

The name of a Highest Key may not be modified under any circumstances.

Searching a file

5.4.2

Inspecting the 'Highest Key' and using the 'READ IN' macro makes searching a File for a specific Record a very simple matter in TAS 2.

Example:

Des	Label	Field 'A'		Field 'B'		Field 'C'		Field 'D'		Field 'E'	
0 5	L 365	COMP	INPUT	WITH	SEARCH	MORE	L 918	EQUAL	L 918	LESS	L 827
0 5	L 827	READ	IN	FROM	INPUT	AT END	L 111				
0 5		GO TO	L 365								

If the File is being updated, the 'WRITE file TO file' macro may be used in exactly the same fashion.

Example:

Des	Label	Field 'A'		Field 'B'		Field 'C'		Field 'D'		Field 'E'	
0 5	L 456	WRITE	MOVEINTO		MOVEOT	AT END	L 123				
0 5		COMP	MOVEIN	WITH	SEARCH	MORE	L 457	EQUAL	L 457	LESS	L 456
0 5	L 457	READ	RECA	FROM	MOVEIN	AT END	L 123				
		etc									

Job Set-up 5.4.3

TAS expects to find all tapes correctly loaded and positioned ready for use. For this reason, the Job Set-up program (C/09/00 for 1" and $\frac{1}{2}$" systems, D/09/00 for $\frac{1}{4}$" systems) *must* always be obeyed immediately before any TAS program involving magnetic tape.

All facilities available through Job Set-up, e.g. Program on Tape, may be used with a TAS 2-compiled program.

On the Job Set-up parameter cards RRN 80 must be set to drum decade 200. RRN 86 must be set to drum decade 272 for the $\frac{1}{2}$" and 1" Tape Control, or to drum decade 244 for the $\frac{1}{4}$" system.

Restarts 5.4.4

All Restarts from specific dumps must be carried out by the Restart routine (C/05/01) as for any other program. The necessary relativiser settings are:-

	I.A.S. word	*Drum word 1 inch and ½ inch systems*	*¼ inch systems*
RRN 75	0	3000	3000
RRN 76	0	2210	2710
RRN 77	390	2200	2700
RRN 80	600	-	2000
RRN 86	-	-	2440

Note that the sprag engaged on the printer will have altered, and that therefore the 'sprag engaged number' in the program will have to be reset. This number is held in I.A.S. location 334.

The Dump Subroutine on the drum will not have been mutilated.

Repositioning 5.4.5

If repositioning (i.e. it is necessary to change tape decks in the middle of a program run) is required in the event of a tape failure, then there must be a special table defined for use by the compiler. Details of this table are given in Appendix Q.