1301 Programmers Manual

Section = Part Five

Original PDF by Roger Holmes
Jan 2009

Sub Sections and HTML version
by Rod Brown

Part 5

I·C·T 1300 SERIES

programmers reference manual

SOFTWARE FACILITIES

Contents

Contents continued Page

Illustration

Part 5

Software Facilities

THE I.C.T. SUBROUTINE LIBRARY 5.1

There is a library of general purpose programs written for the 1300-series computers which are available from I.C.T.

The majority of these routines are subroutines which can be incorporated into the user's own program. There are also programs which are complete in themselves and for which the user need only supply data. Autocodes, which provide facilities for programs to be written in a simpler form than machine-coded instructions, are available and are discussed in more detail later. This part of the manual does not cover every routine in the library but is intended as a guide to the types of routines available.

Index of Library Routines 5.1.1

An index to the routines available is supplied to 1300-series users, together with Specification Sheets that contain sufficient information about each routine to enable the user to select an appropriate routine for his purpose. In some instances a broad class of routines will be the subject of a general paper explaining the best use of each individual routine within that class.

Each individual routine is classified and numbered. The class letter gives the functional division. This is followed by a two-digit number allocated to further identify the function of the routine. The final two-digit number differentiates between routines that have similar functions or achieve the same result in different ways.

For example, a group of routines for printing control is classified as follows:

A/--/--	Indicates an Input/Output routine.
A/02/--	Indicates a print routine.
A/02/06	Prints on one bank only and spaces n lines.
A/02/07	Prints on two banks and spaces n lines.
A/02/08	Prints on three banks and spaces n lines.

Subroutines 5.1.2

A subroutine is a self-contained section of program which may be used more than once in a program, or has a general application and can form part of a number of different programs. In scientific work, for example, the trigonometrical functions sine and cosine, and square roots are required in many routines.

Once programmed as subroutines, any program can include them. In commercial data processing, there is not quite so much scope for generalized subroutines. The P.A.Y.E. calculation is one example of a standard routine that can be incorporated into a program.

On the 1300-series Computers, division has to be programmed and is written as a subroutine. Input and output (P.P.F. programs) are also programmed in subroutine form.

The object, therefore, of a subroutine can be either:-

(a) To save storage space by writing a section of program once only and jumping to that section whenever it is required in the main program, or,

(b) To make programming quicker and easier by incorporating pre-programmed subroutines when available.

Subroutines are useful inasmuch as they can:

(a) Save storage space
(b) Save programming time
(c) Save testing time, as a library subroutine is well planned and fully tested,
(d) Be used time and again with no further effort.

I.C.T. 1300 - series Library Subroutines 5.1.3

The I.C.T. 1300-series Subroutines comprise a collection of well tested routines, which cover:-

Input/Output
Magnetic Tape
Commercial (P.A.Y.E. etc.) Routines
Arithmetical (Division etc.) Routines
Mathematical Routines
Utility Routines
Diagnostic Routines.

Library routines should be used whenever possible for they are proved routines planned to economize in the use of storage space and computer time. By linking them to a main program, the user will reduce considerably the amount of programming and testing time required to achieve successful running of the program.

A Library subroutine complies with a standard format so that a complete subroutine comprises:

Specification Sheet	Flowchart
Program Sheets	Program Card Pack.

Specification Sheets

The following information is shown as fully as possible on all subroutine specification sheets:-

(a) Entry Points

The entry point or points are stated. If more than one, the distinction between them is stated.

(b) Entry Conditions

Locations of variables and parameters assumed by the subroutine on entry are given; also details of the required indicators, and their state.

(c) Results

Results produced by the subroutine together with locations occupied are given.

(d) Storage

The number of words required for the storage of the program and constants is given. Under the heading of temporary storage is shown the number of words used as working space and whose original content is immaterial.

(e) Time

If possible, either exact timings are given or else formulae from which processing time can be calculated.

(f) Limitations

Any limitations in the size of factors and accuracy of results are given.

(g) Error Conditions

A brief account of tests made by the subroutine for any error conditions, together with the action taken if such errors are detected, is given.

(h) Notes

Any other necessary information about the subroutines, not given in other sections, is included here.

Flowcharts and Program Sheets

The specification sheet should normally contain sufficient information for the user to be able to incorporate the subroutine into his program. If, however, further knowledge of the subroutine is required then flowcharts and program sheets can usually be obtained. Some programming points relevant to subroutines are considered below.

Entry to and Exit from Main Program

The instruction to jump from the main program to the first instruction of a subroutine is a normal jump instruction.

For example, either

C	I	D	F	A	R
	10	4	00	0000	17
		--	---	-------	---

or

C	I	D	F	A	R
	21	--	---	-------	---
		4	00	0000	17

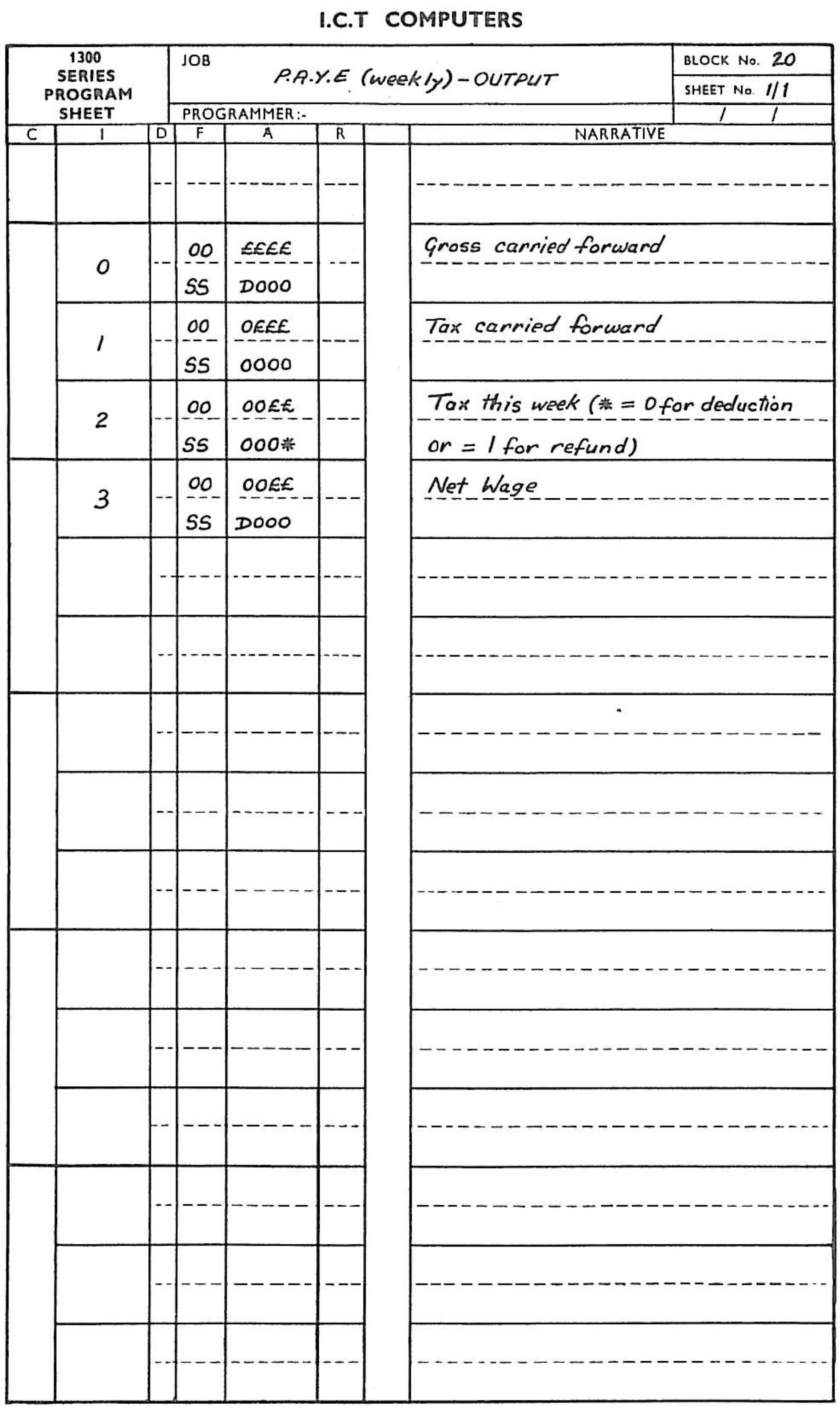
would effect a jump from the main program to the first instruction of a subroutine in I.A.S. location 0 of block 17.

I.C.T COMPUTERS

1300 SERIES PROGRAM SHEET	JOB *P.A.Y.E (weekly) - INPUT*	BLOCK No. *18*
		SHEET No. *1/1*
	PROGRAMMER:-	/ /

C	I	D	F	A	R		NARRATIVE
	0		00	0000			Basic Week Number
			00	00XX			
	1		00	0000			* = 0 for Normal
			00	000*			or = 1 for Week 1 Designation
	2		00	0000			Number of Holiday Weeks
			00	000X			
	3		00	00££			Week 1 Free Pay
			SS	0000			
	4		00	££££			Gross Wage brought forward
			SS	D000			
	5		00	00££			Gross Wage this week
			SS	D000			
	6		00	0£££			Tax brought forward
			SS	0000			
	7		00	00££			Total Deductions
			SS	D000			

Figure 49: INPUT INFORMATION



I.C.T COMPUTERS

1300 SERIES PROGRAM SHEET	JOB P.A.Y.E (weekly) - OUTPUT	BLOCK No. 20
		SHEET No. 1/1
	PROGRAMMER:-	/ /

C	I	D	F	A	R		NARRATIVE
	0		00	££££			Gross carried forward
			SS	D000			
	1		00	0£££			Tax carried forward
			SS	0000			
	2		00	00££			Tax this week (✱ = 0 for deduction
			SS	000✱			or = 1 for refund)
	3		00	00££			Net Wage
			SS	D000			

Figure 50: RESULTS OF CALCULATIONS

I·C·T 1300 SERIES

programmers reference manual

SOFTWARE FACILITIES

Contents

Contents continued Page

Illustration

Part 5

Software Facilities

THE I.C.T. SUBROUTINE LIBRARY 5.1

There is a library of general purpose programs written for the 1300-series computers which are available from I.C.T.

The majority of these routines are subroutines which can be incorporated into the user's own program. There are also programs which are complete in themselves and for which the user need only supply data. Autocodes, which provide facilities for programs to be written in a simpler form than machine-coded instructions, are available and are discussed in more detail later. This part of the manual does not cover every routine in the library but is intended as a guide to the types of routines available.

Index of Library Routines 5.1.1

An index to the routines available is supplied to 1300-series users, together with Specification Sheets that contain sufficient information about each routine to enable the user to select an appropriate routine for his purpose. In some instances a broad class of routines will be the subject of a general paper explaining the best use of each individual routine within that class.

Each individual routine is classified and numbered. The class letter gives the functional division. This is followed by a two-digit number allocated to further identify the function of the routine. The final two-digit number differentiates between routines that have similar functions or achieve the same result in different ways.

For example, a group of routines for printing control is classified as follows:

A/--/--	Indicates an Input/Output routine.
A/02/--	Indicates a print routine.
A/02/06	Prints on one bank only and spaces n lines.
A/02/07	Prints on two banks and spaces n lines.
A/02/08	Prints on three banks and spaces n lines.

Subroutines 5.1.2

A subroutine is a self-contained section of program which may be used more than once in a program, or has a general application and can form part of a number of different programs. In scientific work, for example, the trigonometrical functions sine and cosine, and square roots are required in many routines.

Once programmed as subroutines, any program can include them. In commercial data processing, there is not quite so much scope for generalized subroutines. The P.A.Y.E. calculation is one example of a standard routine that can be incorporated into a program.

On the 1300-series Computers, division has to be programmed and is written as a subroutine. Input and output (P.P.F. programs) are also programmed in subroutine form.

The object, therefore, of a subroutine can be either:-

(a) To save storage space by writing a section of program once only and jumping to that section whenever it is required in the main program, or,

(b) To make programming quicker and easier by incorporating pre-programmed subroutines when available.

Subroutines are useful inasmuch as they can:

(a) Save storage space
(b) Save programming time
(c) Save testing time, as a library subroutine is well planned and fully tested,
(d) Be used time and again with no further effort.

I.C.T. 1300 - series Library Subroutines 5.1.3

The I.C.T. 1300-series Subroutines comprise a collection of well tested routines, which cover:-

Input/Output
Magnetic Tape
Commercial (P.A.Y.E. etc.) Routines
Arithmetical (Division etc.) Routines
Mathematical Routines
Utility Routines
Diagnostic Routines.

Library routines should be used whenever possible for they are proved routines planned to economize in the use of storage space and computer time. By linking them to a main program, the user will reduce considerably the amount of programming and testing time required to achieve successful running of the program.

A Library subroutine complies with a standard format so that a complete subroutine comprises:

Specification Sheet	Flowchart
Program Sheets	Program Card Pack.

Specification Sheets

The following information is shown as fully as possible on all subroutine specification sheets:-

(a) Entry Points

The entry point or points are stated. If more than one, the distinction between them is stated.

(b) Entry Conditions

Locations of variables and parameters assumed by the subroutine on entry are given; also details of the required indicators, and their state.

(c) Results

Results produced by the subroutine together with locations occupied are given.

(d) Storage

The number of words required for the storage of the program and constants is given. Under the heading of temporary storage is shown the number of words used as working space and whose original content is immaterial.

(e) Time

If possible, either exact timings are given or else formulae from which processing time can be calculated.

(f) Limitations

Any limitations in the size of factors and accuracy of results are given.

(g) Error Conditions

A brief account of tests made by the subroutine for any error conditions, together with the action taken if such errors are detected, is given.

(h) Notes

Any other necessary information about the subroutines, not given in other sections, is included here.

Flowcharts and Program Sheets

The specification sheet should normally contain sufficient information for the user to be able to incorporate the subroutine into his program. If, however, further knowledge of the subroutine is required then flowcharts and program sheets can usually be obtained. Some programming points relevant to subroutines are considered below.

Entry to and Exit from Main Program

The instruction to jump from the main program to the first instruction of a subroutine is a normal jump instruction.

For example, either

C	I	D	F	A	R
	10	4	00	0000	17

or

C	I	D	F	A	R
	21				
		4	00	0000	17

would effect a jump from the main program to the first instruction of a subroutine in I.A.S. location 0 of block 17.

The subroutine is so written that, when it has been completed, it automatically restores control to the instruction in the main program immediately following the jump to the subroutine.

Storage and Relativizers

Storage addresses written relative to the first location of a subroutine will use block relativizer B. Temporary storage will use relativizer 1. If other relativizers are needed by the subroutine (e.g. for a block of data), relativizers 3 to 9 will be used.

Indicators

Unless specified as an entry condition, library subroutines make no assumptions about the state of indicators on entry to the subroutines, and users of subroutines should make no assumptions about the states of indicators used by subroutines when re-entering the main program.

Programmed indicators employed by subroutines will be used in the reverse order 19, 18, 17....

Library subroutines will not test the I.A.S. parity indicator unless this is stated on the specification sheet.

It is important to note that Library subroutines can be assumed to be resetting unless a definite statement to the contrary appears on the specification sheet.

AUTOCODES 5.2

An autocode enables a programmer to write a program in a symbolic language which uses alphabetic names and characters instead of numeric references. In some autocodes this use of alphabetic names enables the program to be written in comprehensible English. A special program is then employed to translate or interpret the autocode language program into machine coding before it is obeyed by the computer. Thus the autocdoe method of programming is designed to:

(a) enable programs to be written in less time,

(b) reduce the amount of writing (and similar clerical work), and therefore the chance of error with a consequent reduction in testing time,

(c) simplify program maintenance,

(d) make computer procedure comprehensible to people not trained in the machine code and to enable these people to prepare programs with the minimum amount of training,

(e) standardize coding tactics and therefore permit the exchange of ideas and programs among computer users.

When writing a program in autocode, the programmer does not have the same control over the instructions or positions of storage as when writing a machine-coded program. Consequently the programmer cannot adapt the program to suit his own requirements ideally. Thus a program written in autocode is not usually quite as efficient in time or storage as a program written in machine code. This disadvantage of autocode program efficiency is usually more than offset by the advantages listed above.

Initially, a source program is written in the autocode, which is a restricted form of language conforming to certain defined rules. The source program must then be transformed into a machine -coded program: this machine-coded program is termed the object program. The transformation is done by the computer under the direction of a processor program.

The processor program may be either an interpreter or a translator. A translator transforms the entire source program into the machine-language object program; the resultant object program is then stored in its entirety and is either output in the form of punched cards or obeyed immediately.

If the object program is obeyed immediately the complete translation of the source program is achieved, then the system is known as a load-and-go system. The term load-and-go is derived from the fact that the complete process of translation and execution of the translated program is achieved in a single run on the computer.

An interpreter transforms the source program into machine-coded instructions and each machine-coded instruction is obeyed immediately it has been created. Thus no complete object program is produced and is therefore not available as output.

The programming language may be a machine-oriented language or a procedure-oriented language.

A processor for a machine-oriented language is termed an assembly system or assembler. A machine-oriented language is devised for a particular type of computer and consists of a code which is easier to learn than machine code. Programs can also be written more quickly using an assembly language than using machine code. There is, however, a direct connection between the two, and one source program instruction often results in one machine-coded instruction in the object program. An exception to this arises for input and output which is concisely achieved using an assembly system.

A processor for a procedure-oriented language is termed a compiler. A procedure-oriented language is not necessarily devised for a particular computer. The same language may be used for several different types of computer, each computer having its own compiler for converting the source program into the appropriate machine-coded program. The source program for a compiler resembles a restricted form of language which conforms to the rules laid down for the autocode. A compiler is more powerful than an assembler and one statement in a procedure-oriented language usually results in several machine-coded instructions in the object program.

Further, an assembly system or compiler will be oriented towards the application to which the source programs refer. For example, Rapidwrite is a commercially-oriented language and MAC (Manchester AutoCode) is a scientifically and mathematically-oriented language.

There are at present four autocodes recommended for use on I.C.T. 1300-series Computers. They are

TAS	(Thirteen-hundred Assembly System)	***Rapidwrite***	
MPL	(Mnemonic Programming Language)	***MAC***	(Manchester AutoCode)

Brief introductory details of these four autocodes are given in the following paragraphs and a reference can be made to the appropriate manuals, which give full details.

The use of these autocodes is dependent upon certain minimum machine requirements (such as the size of I.A.S. and magnetic drums etc.). Thus several variations of each processor may exist. The specification sheets for the I.C.T. Subroutine Library Routines give full details of machine requirements and storage.

Rapidwrite 5.2.1

Rapidwrite is a commercially-oriented autocode. The source program is converted to machine code by a translator/compiler program.

Initially, a source program is prepared using specially designed pre-printed forms and dual -purpose cards. The latter enable the program to be built up on the lines of a flowchart with moveable blocks. The fixed format of the dual-purpose cards and the forms reduces the amount of writing and punching and, in consequence, errors resulting from these operations are also reduced. The source program is read into the computer and a preliminary routine produces a full descriptive print-out of the program in English. When the program has been corrected by using this routine, it is again read into the computer and the object program is compiled. The object program will be both punched and printed. The print-out of the object program will contain a reference to the print-out obtained from the preliminary routine. The object program will be ready for testing after insertion of any required subroutines and the addition of a standard program.

A Rapidwrite source program is written in three clearly defined divisions: Environment, Procedure and Data divisions.

The Environment division, which is filled in on a pre-printed form, deals with the specification of the computer and includes such data as the size of I.A.S. and drum storage and console indicators which are to be used.

The Procedure division deals with instructions handling the processing of the problem and is written on special dual-purpose cards and one card is allocated for each instruction such as READ, WRITE, COMPUTE, IF or GO.

The Data division which deals with the organization of data is written on special forms. Data names not exceeding five characters in length are used and if required these can be translated into full data names, up to 30 characters in length, by supplying the processor with a list showing the full names against their abbreviations. The data form when complete will contain all the relevant information on the input and output files required by the processor for the storage of data, the kind of information held in a field and the allocation of working storage.

TAS 5.2.2

TAS (<u>T</u>hirteen-hundred <u>A</u>ssembly <u>S</u>ystem) is a commercially-oriented autocode and is provided as an intermediate language which lies between machine language and a full autocode language such as Rapidwrite. Thus, a programmer writing in TAS will require a greater knowledge of the machine code than one using Rapidwrite. The source program is converted to machine code by a translator/assembler program. Each TAS instruction generates an *average* of four machine code instructions and thus writing time is considerably reduced. TAS is ideal in cases where programs are needed urgently or for small programs e.g. those requiring less than two weeks machine coding.

The source program is written in TAS language on special stationery from which one card is hand punched for each line of entry. These cards are then fed to the computer and the object program is produced on fast-read program cards in one run. A print-out of the source program and other print-outs to aid the programmer can also be produced.

To write a TAS program, it is necessary to:

(a) Specify tables for storage on the drum.

(b) Describe each Input card.

(c) Describe each Output line.

(d) Describe each Output card.

(e) Write the procedure program from a selection of functions with the facility of adding subroutines and machine-coded instructions.

Data read from a card or printed on a line are referenced in the program as a field name, the name being introduced when the programmer describes each card or line of print. Data of a bulky or lengthy nature are best handled in the form of tables which are stored on the drum.

There is an assembler available which allows the programmer to use magnetic-tape storage and special instructions are included in the autocode for controlling the reading and writing of tape. Facilities are provided for the programmer to specify his own tape records, each record being described as a series of field names.

MPL 5.2.3

MPL (Mnemonic Programming Language) is a commercially-oriented language specially designed for users with a small machine configuration which does not permit the use of TAS or Rapidwrite etc. MPL may be used on a machine with the basic configuration of 400 words I.A.S. and a 3,000 word drum. The MPL processor may be used as a load-and-go assembler or may punch out the object program, the appropriate mode of operation being selected by means of a switch.

The source program is written in the MPL language which has a one-character function code and five-character operand. Most MPL instructions result in a corresponding machine-code instruction being generated i.e. on a one-for-one basis. The main exceptions to this are the powerful input/output and division macro-instructions.

When the object program has been created, it is stored automatically by the assembler. When the object program has been stored, the assembly program is overwritten by a standard control program pack which contains an input/output package. This package may incorporate either a P.P.F. control routine or a routine for serial batch processing.

When the standard control program pack has been read in, it is possible to run the program, for it is not necessary to punch out the object program before testing takes place. When the object program has been satisfactorily tested, a proved object program may be punched on cards.

MAC 5.2.4

MAC (Manchester AutoCode) is a mathematically and scientifically-oriented autocode and has been designed for programs which comprise the solution of mathematical equations or formulae.

The MAC compiler operates on the load-and-go principle only. Thus the object program is compiled each time a job is run. This is not however significant since the time taken to compile the object program is small. Further, since the problems for which MAC is used are of a mathematical nature, the object program is not normally run at regular intervals as is a commercial program.

A MAC source program comprises autocode instructions and directives. Usually a MAC instruction will result in several machine-coded instructions being produced: these instructions forming the object program.

Directives transmit information to the translator for allocation of storage and do not form part of the object program.

The translator converts MAC instructions in the source program to machine-coded instructions which will produce for example, the procedure to solve an equation. A trigonometrical or logarithmic function may be initiated in MAC using a single instruction.

P.P.F. PROGRAMS AND INPUT/OUTPUT ROUTINES 5.3

P.P.F. Programs 5.3.1

It is usual for the computer to process jobs that involve the use of the card reader and punch and the printer. In general, one card being read will not result in one card being punched or one line being printed. Assume, for a practical example, that the duty cycle consists of five cards read, one card punched and six lines printed. This task could be tackled serially, as follows:-

Program to read five cards
Program to process data
Program to punch one card
Program to print six lines.

On a machine with a 600 cards a minute reader, 100 cards a minute punch and 600 lines a minute printer the time taken to read five cards is 500 milliseconds; the time taken to punch one card is 600 milliseconds and the time taken to print six lines is 600 milliseconds. Consequently, the input/output share of the duty cycle takes 1.7 seconds. If the three input/output programs were integrated into one program, so that the time taken was the time of the longest operation, i.e. 600 milliseconds, far greater efficiency would result and the task would be tackled in parallel as follows:-

Program to read five cards punch one card and print six lines

Program to process data.

The punch and the printer are, of course, operating on output data from the previous processing.

The peripheral units are cyclic in operation in that, having produced or accepted one unit of information, they will not produce or require another unit of the information until a set time interval has elapsed. The program dealing with this unit of information may not take as long as this interval and there may be spare time that can be used by the other units.

Therefore, integration is possible without taking any additional time over that taken for the longest single operation.

Print, Punch and Feed (P.P.F.) programs are written as three separate programs - a card reader program, a card punch program and a line print program linked together by means of a control routine which ensures that the priority of the different programs is in accordance with the requirements of the input/output units. With a P.P.F. in operation there is no further spare time in which to process data, so that duty cycle is:-

P.P.F. - Process.

The main program sets certain keys before entering a P.P.F. These keys instruct the P.P.F. as to how many cards are to be read and punched and how many lines are to be printed. These keys are not necessarily constant and may vary throughout a program to permit, for example, the number of lines of print produced to be different each time the P.P.F. is entered. Several comprehensive P.P.F. programs are available from the I.C.T. Subroutine Library.

As card readers and line printers with differing speeds are available, the most efficient duty cycle and the timings will be different for different machines. The example given above applies to a machine with a 600 cards a minute reader and 600 lines a minute printer. For a 300 cards a minute reader and 300 lines a minute printer, the variables could be as follows:-

Duty cycle:-	3 cards read	Time to read 3 cards is 600 ms
	1 card punched	Time to punch 1 card is 600 ms
	3 lines printed	Time to print 3 lines is 600 ms.

Thus the input/output share of the duty cycles is 1.8 seconds but using P.P.F. the time taken will be 600 milliseconds.

Input/Output Routines 5.3.2

Besides the P.P.F. routines, there are also Library routines available for the individual control of each peripheral unit. Some of these provide an exit from the subroutine to enable the programmer to time-share some of his main program with the peripheral unit program.

General routines are also available which distribute data into the format required by the output routines, or alternatively distribute information read by an input routine into a form suitable for processing. The positions of the data fields for processing are communicated to the distribution routine by keys which are set by the main programmer.

MAGNETIC-TAPE CONTROL AND UTILITY ROUTINES 5.4

Tape Layout 5.4.1

The layout and organization of data on magnetic tape should conform to an I.C.T. standard format. Essentially, data are recorded on tape in blocks and the length of a block is defined by an end of block marker; see 3.8.1. Data within blocks can be grouped into fixed- or variable-length records. It is the programmer's responsibility to arrange data into conveniently sized records and blocks.

There are three main considerations in tape layout:

(a) Tape Labels
(b) Block Layout
(c) Record Layout.

Tape Labels

Two identification labels must be written to tape; these are the beginning of tape label and the end of tape label.

The beginning of tape label precedes the first data block. This label contains information by which the reel can be identified, together with a date at which the tape can be overwritten and a count of the number of times the tape has been written. The standard format for the beginning of tape label is described in the Tape Housekeeping Manual.

The use of beginning of tape labels ensures that the correct tapes have been fitted and prevents master information being overwritten if a wrong tape is loaded.

The end of tape label must appear at the end of every tape reel. For a single reel file the end of tape label is an end of file label. However, if a file comprises several reels, all but the final reel must bear an end of reel label. The end of tape label contains a count of the number of blocks in the reel. The standard layouts for the end of reel and end of file labels are described in the Tape Housekeeping Manual.

Other than the beginning of tape label block, all label blocks are short blocks, i.e., blocks of exactly four words, including the end of block marker.

Block Layout

A block will consist of a number of sorted or unsorted records. Each record is identified by a key. The first word (word 0) of a block consisting of sorted records must contain the most

-significant word of the key of the last record in the block (i.e. the highest key). If a block consists of unsorted records, the contents of the first word are immaterial.

Word 0 is followed by the first record in the block.

The last word of the block is the end of block marker i.e. a word of $\overline{15}$s.

Example

```
                                   INTER-BLOCK GAP
Word 0          1st word of key of record number 3
Word 1          007 XXX XXX XXX  ⎫
   .                             ⎬  1st record
Word 3          1st word of key  ⎪
Word 4          2nd word of key  ⎭
   .
   .
Word 8          025 XXX XXX XXX  ⎫
   .                             ⎬  2nd record
Word 10         1st word of key  ⎪
Word 11         2nd word of key  ⎭
   .
   .
Word 33         016 XXX XXX XXX  ⎫
   .                             ⎬  3rd record
Word 35         1st word of key  ⎪
Word 36         2nd word of key  ⎭
   .
   .
                __ __ __ __ __ __ __ __ __ __ __ __
Word 49         15 15 15 15 15 15 15 15 15 15 15 15

                                   INTER-BLOCK GAP
```

Record Layout

The three most-significant digits of the first word of each record must contain the number of words in that record.

The key, identifying the record, may consist of either alphabetic or numeric data. If an alphanumeric key is required to be used during sorting, then the data must conform to the ZNZNZN ----ZN format (where Z = Zone and N = Numeric), i.e. the code components for each character must be adjacent. The key may consist of any number of consecutive words (with the exception that some routines, sorting for example, limit the number of words in a key) and the first word of the key may be located anywhere in the record. The position of the first word of the key within the records of one file must be constant. For example in the illustration above, two words precede the first word of the key in each record. The number of words preceding the first word of the key will be a parameter that has to be given to Tape Sorting and other routines. Keywords must have values in the range of 000 000 000 001 to 499 999 999 999 inclusive.

Tape Housekeeping Routines 5.4.2

I.C.T. provide programs to cover all aspects of reading data from, and writing data to, magnetic tape. This series of programs is known collectively as Tape Housekeeping Routines. A manual is available that gives full details of these routines.

Housekeeping routines are provided to simplify the using of magnetic tape by assuming control of all reading, writing and associated contingencies. These contingencies include correction of errors, treatment of end of reel, end of file and beginning of reel conditions.

The routines are usually in the form of program packages, that is, a number of interdependent programs controlling all the various aspects of reading or writing tape.

For example a Tape Write Package is a package consisting of programs to facilitate:

Tape Preparation

Write Label Check

Tape Write

Write Exceptions.

Briefly, these programs cover such contingencies as:

(a) checking the initial setting-up of the appropriate deck by testing for mechanical readiness,

(b) testing for presence or absence of a writing ring,

(c) the checking and identification of labels,

(d) the writing of tape,

(e) error detection and correction, end of tape condition and short blocks etc.

Packages are available for writing and reading tape and also for controlling both reading and writing.

A full list of Tape Housekeeping routines available will be found in the Subroutine Library Index of Specification Sheets.

It should be noted that certain conventions are observed in the writing and subsequent using of the Housekeeping routines. These conventions are described fully in the Tape Housekeeping Routines Manual.

Job Set-up

The Job Set-up routine is employed at the commencement of any job using magnetic tapes and ensures that the correct tapes have been fitted for the job and that the tape decks are mechanically ready. The routine will create labels on write tapes, check labels on read tapes, and set up areas of information on the drum and in I.A.S. without which other Tape Housekeeping routines will not function correctly. Thus, this routine will ensure that the tape decks are correctly loaded and ready for a job to be run. On exit from this routine, the tapes are positioned for writing or reading after the beginning of tape label. If the user program is held on tape this routine will load that program on the drum.

Write Program to Tape

A program stored on the drum may be written to tape by using two Subroutine Library routines: 'Write Program to Tape' and a selected write to tape program. The former routine prepares a program held in absolute form on the drum for writing to tape and the selected write routine enables this program to be written to tape.

The Write Program to Tape routine performs no tape operations but prepares the program to be written to tape from the drum in the following manner:

(a) The spurious end of block markers are removed i.e. any constants consisting of words of $\overline{15}$s which are not intended as end of block markers.

(b) The program is arranged in contiguous locations.

(c) Standard program labels are manufactured.

(d) The program is arranged in blocks of a specified size.

An exit is made to the user's write routine for the writing of each block.

Magnetic-Tape Sorting

Several routines which facilitate the sorting of data on magnetic tape are available from the Subroutine Library. These routines usually require the use of four or three tape decks or three tape decks and a magnetic drum.

Merging Using Four Tape Decks

The most fundamental method of sorting is the merge. In this routine records are written from two input tapes, formed into pairs in sequence and read to two output tapes. This sequence of operations is known as a pass. On the next pass, the tapes reverse their rôles, that is, the tapes that were used previously for output are now used for input and vice versa. The pairs of records are merged to form groups of four and on the next run groups of eight and so forth, the length of the group being doubled on each pass. A group of records in sequence of this nature is known as a string. The merging is continued until all the records form one string. It will require $\log_2 N$ passes, where N is the number of records to be sorted, the logarithm being taken as the next highest integer.

In order to save computer time, before entering the merge routine, the records are normally partially ordered (using a special pre-stringing routine) into strings of ordered records. These strings are each an integral number of blocks. The pre-stringing routine arranges the blocks from one tape onto two tapes ready for input to the merge routine. The merging sort then requires $\log_2 S$ passes, where S is the number of strings.

Merging Using Three Tape Decks

A Tape Record Merge using three tape decks operates in basically the same manner as that required for the four tape deck version. Merging takes place between two input tape decks, the resultant strings being written on a third tape deck. The next pass reads the strings from the third tape deck so that the strings are written alternately to the two tape decks initially employed for input. Thus two tapes are prepared for a subsequent merge pass. Merging using three tape decks doubles the number of passes required to produce a completely sorted tape by using four tape decks.

This routine should be used in preference to the Three Tape and Drum Sort described below when large volumes of data are involved.

Sorting Using Three Tape Decks and the Magnetic Drum

If part of the magnetic drum is available for use in the sort, a routine is available so that sorting may be achieved by the following method.

Initially a batch of records are read from deck address 1 to the magnetic drum; see Figure 51.

These records are then sorted on the drum and written to tape on deck address 2. On the next pass a second batch of records are read from deck address 1 to the magnetic drum, sorted internally and then merged with the records on deck address 2 to deck address 3.

Subsequent merging is from deck address 3 to deck address 2, then back again until the whole of the input file has been sorted.

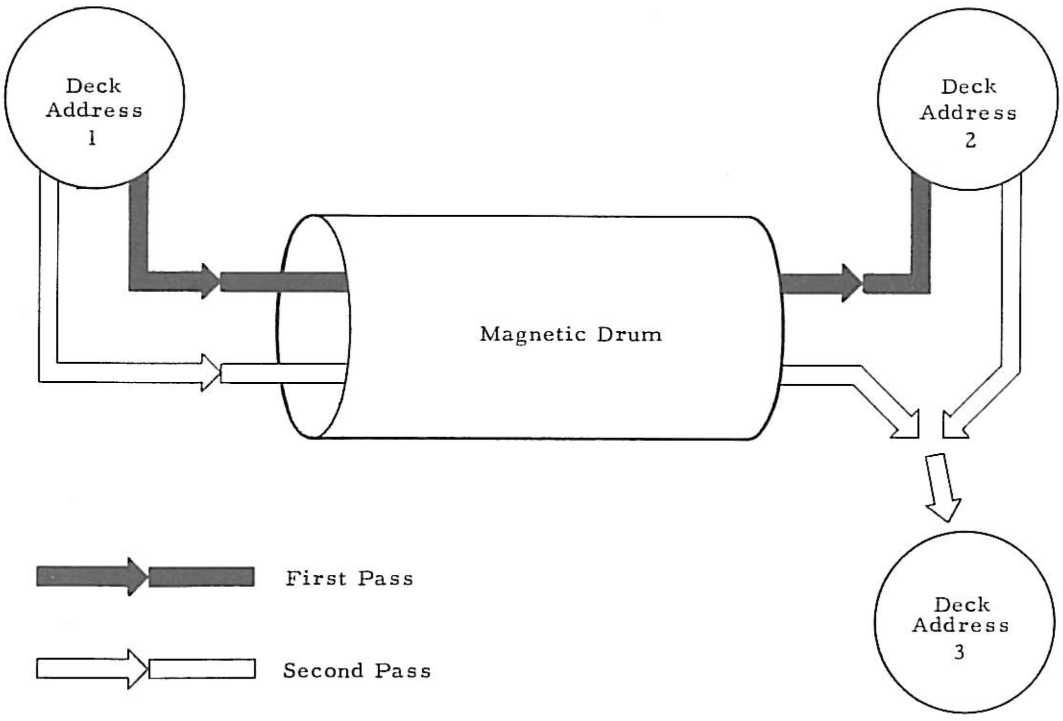


Figure 51: SORTING USING THREE TAPE DECKS AND THE MAGNETIC DRUM

GENERATOR ROUTINES 5.5

A generator is a routine that, when provided with parameters, creates instructions to form a routine which the programmer requires. Generators are available for creating input and output distribution routines and drum sorting routines.

A generator is provided with keys similar to those provided for a general program. For example, for an input distribution program, the keys specify the position into which the data fields are to be distributed. The generator differs from the general routine in that having extracted the required information from the keys, it does not obey the routine but punches it on cards. The punched routine can then be included in the main program as a subroutine.

It is likely that, in a given job, the keys for a general routine will be the same every time it is used. In this case, it is preferable to use a generator and include the generated program in the main program rather than the general routine. This will save the computer time used by a general routine to extract the same information from keys every time the program is run.

DIAGNOSTIC ROUTINES 5.6

A diagnostic routine is a routine which has been designed to locate an error in programming; i.e. to determine where in a faulty program a program error has occurred.

The testing of a program by using diagnostic routines is accomplished in what may be considered as three basic steps:

(a) Checking the validity of data punched in the program cards.

(b) Checking storage of the program.

(c) Testing the program itself, in sections.

I.C.T. diagnostic routines are available from the Subroutine Library to facilitate the testing of programs as outlined above.

Validity Check 5.6.1

With the aid of the I.C.T. Validity Check routine, a validity check may be made on the program pack to ensure that all the instructions in a program conform to the correct layout required for the machine specification; any instructions which are in error (i.e. they do not conform with the correct format) are printed out. Enough information is printed out to enable such errors to be located and corrected. This routine ensures that if errors arise in a program, it is not due to incorrect program instructions (instructions not in the correct format) being read in.

Proving Storage 5.6.2

A diagnostic routine, called the I.A.S. and Drum Print-out routine is available. This routine can be used for obtaining a complete print-out of the program as it is stored on the relevant area of the drum. This diagnostic routine will enable any incorrectly set relativizers to be detected.

Memory Dump 5.6.3

The examination of a program during a test run may be achieved by using the I.C.T. Memory Dump routine. This diagnostic routine enables the contents of I.A.S. and certain drum channels to be automatically printed out at specified points in the program.

The routine is read into the machine with the program under test together with certain specifications. These specifications will state the following:-

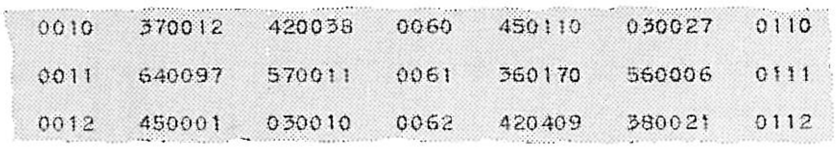
(a) The exact location of points in the program under test at which certain areas of storage are to be printed out.

(b) Which groups of drum channels are to be printed out. One group of drum channels to be printed out is specified for each break-in point in the program under test.

Essentially, the Memory Dump routine may be considered as being in two sections - a preliminary routine and a print-out routine. Control is initially transferred to the preliminary routine (by arrangement of card packs) so that this routine, by using the specifications in (a) above, can substitute into the program under test, instructions to facilitate the entry into the print-out at the required points. The program under test is then run normally on test data as if the Memory Dump routine were not present. At the required points in the program under test, the substituted instructions cause entry to the print-out routine and all the contents of I.A.S. are printed out together with the specified drum channels.

The program under test then resumes control and continues to run in the normal manner.

The print-out will appear in blocks of two hundred words these being fifty lines of four words to a line, e.g.



```
0010   370012   420038   0060   450110   030027   0110
0011   640097   570011   0061   360170   560006   0111
0012   450001   030010   0062   420409   380021   0112
```

The above is an example of a segment of such a print-out.

It should be noted that the original state of the program under test is restored when the print -out routine is entered, thus the instructions referring to the Memory Dump routine are not printed and no space need be left in I.A.S. for the Memory Dump when writing the program under test.

The Memory Dump routine enables the programmer to determine which sections of program contain errors. To find the *exact* point in a program which is in error a Trace routine must be employed.

Trace Routine 5.6.4

To pin-point an error in a section of a program it is of course possible to step through the section of program manually on the computer observing the contents of registers etc., but this involves the cost of much machine time. The same results may be obtained much more efficiently with the I.C.T. Trace routine. This diagnostic routine is a great aid to the programmer in testing programs. It produces:

(a) A print-out of the instructions obeyed by the program under test in the order in which they were obeyed.

(b) A print-out of other relevant information about the effects produced by these instructions.

The Trace routine is used in much the same way as the Memory Dump routine, the main differences being in the specifications given with the Trace routine and the information printed out.

The specifications give the start and end points of the sections of the program under test which it is desired should be traced. When the program under test reaches the substituted instructions to enter the trace (substituted by a preliminary routine as in the Memory Dump) and the trace print-out is called in, the trace obeys the program under test one instruction at a time. After each instruction has been executed the following information is printed out:

(a) The I.A.S. location of the instruction just obeyed.

(b) The instruction just obeyed.

(c) The contents of Register B.

(d) The contents of the I.A.S. location specified in the instruction.

(e) The time which the program would have taken to reach this point.

An example of a segment of a trace print-out shown below.

I.A.S. Location of Instruction obeyed	Designation	Function	Address	Contents of Register B after obeying current instruction	Contents of I.A.S. referred to after obeying current instruction	Time Accumulated so far (Microseconds)
0001	0	67	0019	000000000101	000000000001	375
	4	02	0001	000000000101		
0001	0	67	0019	000000000101	000000000000	412
	4	02	0001	000000000101		
0002	0	57	0011	000000000000	000000000000	453
	0	42	0001	000000000000	000000000000	474
0003	0	37	0002	570011420001	570011420001	507
	0	36	0003	770013760003	770013760003	528
0004	0	62	0003	540027520006	770013760003	561
	0	40	0004	540027520006	000000000000	582
0005	4	00	0004	540027520006		
0004	0	00	0000	540027520006	000000000000	606
	0	00	0000	540027520006	000000000000	618
0005	4	00	0004	540027520006		
0004	0	00	0000	540027520006	000000000000	642
	0	00	0000	540027520006	000000000000	654

The information displayed in the print-out should allow program errors to be easily located.

Other trace routines with special characteristics are available, which may be used in preference to the full I.C.T. Trace routine.

If full details provided by the I.C.T. Trace routine are not required, the Indicator Trace routine might be used. This routine operates under the control of a manual indicator and prints either as the full trace or else suppresses printing on all except jump instructions (i.e. successful indicator tests).

The I.C.T. Trace routine slows down the rate at which instructions in the program under test are executed. Thus the I.C.T. Trace routine cannot be run on sections of a program which contain input, output or magnetic-tape operations. If the program uses standard proven library subroutines, there is no need to waste time tracing them.

The Trace routine can therefore be obtained in such a form that subroutines which are not required to be traced can be automatically avoided.

Program Updating Routine 5.6.5

Having traced an error in the testing of a program, it then becomes necessary to rectify the programming error.

The amendment of the program by means of the insertion or deletion of words, besides creating the problem of re-allocation of storage, causes other difficulties. Jumping over an unwanted portion of program is wasteful of storage and untidy. Jumping out to previously unused storage locations in order to effect insertions can cause great confusion as the number and complexity of amendments increases. The actual deletion of a section of program or insertion of a group of words, besides making a rewriting of the particular block of program necessary, invalidates addresses elsewhere in the program. These addresses could be located and converted but the labour involved could be immense.

The I.C.T. Program Updating routine is designed to reduce this labour. If amendments have been made by inserting or deleting words throughout the program card pack and resetting the relativizer settings to the re-allocated storage locations, then the program pack can be fed as data cards to the Program Updating routine.

The Program Updating routine pack, together with specifications indicating the size and location of deletions and insertions which have already been made to the program under test is read into the machine. The program under test is then placed in the card read hopper and is read and processed by the Program Updating routine. Depending on the setting of manual indicators, the routine produces a printed and/or punched version of the program under test in which all addresses made invalid by the insertions or deletions have been corrected. The printed output is in the format of a normal program giving the absolute-address form of the program alongside the relative-address form.

UTILITY ROUTINES 5.7

Routines are available that do not belong to any particular category but have such general applications that they may be included in almost any type of program. These Utility routines cover such programs as division routines, sorting routines and routines for detecting parity errors and taking appropriate action.

Division Routines 5.7.1

Division is not built into the hardware of the computer but is accomplished in subroutines by repeated subtraction. Routines are available which cover both decimal and sterling division for positive and negative numbers.

Parity Error Routines 5.7.2

Various routines are available for testing the drum and I.A.S. parity indicators after a drum transfer has been obeyed. The programming implications of parity errors are described in 4.8. It should be noted that:

(a) An I.A.S. parity check is made when data are transferred from I.A.S. Hence an I.A.S. parity error can only be detected when the transfer is from I.A.S. to the drum. If the I.A.S. parity indicator is found to be set then the computer is brought to a stop (11 1006 displayed in CR3). If this stop is encountered then a return must always be made to the previous restart point.

(b) A drum parity check is made when data are transferred from the drum. Hence a drum parity error can only be detected when the transfer is from the drum to I.A.S. If the drum parity indicator is found to be set then the computer is brought to a stop with 11 1007 displayed in CR3. When the Start button is pressed a further attempt is made to obey the drum transfer. If there is a persistent failure, then a return must be made to the previous restart point.

(c) A parity error routine (or a programmed test of indicator 07) should be used for *every* transfer from the drum. If a parity failure is detected on a transfer from the drum which is not followed by a test of indicator 07 then the program will detect this error the next time indicator 07 is tested. Indicator 07 will be unset by testing and a repeat of the drum transfer preceding the indicator test (which is not the transfer in which the parity check failed) will apparently correct the error.

For some of the library routines the drum transfer is given as a parameter and is obeyed in the subroutine. It is recommended that these should be used since they make it possible for a drum transfer instruction to overwrite its own storage location in I.A.S. If a drum parity error is detected then, as the drum transfer is retained in the subroutine, another attempt can be made to effect the transfer.

Zero Suppression Routines 5.7.3

During output distribution, particularly prior to printing, it is very often necessary to distinguish between significant and non-significant digits in a data field. For example, if a quantity is allocated six positions on the printed sheet, and a particular value is only a four-digit number, then it is likely that it will be required to suppress the two non-significant zeros (leaving spaces) and print only the four-digit number. This process is known as zero suppression. It consists of giving a zone component of zero to non-significant zeros and a zone component of one to significant digits. Standard zero suppression routines are available for both decimal and sterling fields.

Punching Program into Fast-read Cards 5.7.4

Fast-read program cards (preceded by a control word of designation F) enable five 12-column words per program card to be read in under Initial Orders. Each word is punched exactly as it appears within the computer i.e. all addresses are absolute, negative numbers are in complementary form etc. Thus, by using fast-read cards, Initial Orders merely read and store the data read in the form in which they are punched. A full description of Initial Orders and fast-read cards will be found in the Initial Orders Manual.

Fast-read cards may be punched in the correct format from data held on the magnetic drum, i.e. a proved or compiled program may be punched into fast-read cards from the drum for later use as input.

Several subroutines are available from the Subroutine Library which enable fast-read cards to be punched in the required format from the absolute information as it is held on the drum.

Sorting Routines 5.7.5

The arrangement of data records into ascending numeric sequence, is an essential part of computer usage. The sequence is arranged according to a key held in a given position within each record. Sorting methods are available which enable source items to be examined in their initial sequence and thence to be rearranged in a required sorted sequence. Many subroutines are available from the Subroutine Library which cater for most aspects of sorting of fixed-length or variable-length records. Data held on the drum or in I.A.S. may be sorted and several sorting methods are available. The present subroutines available provide three basic methods of sorting data in I.A.S. and on the drum; these methods are:

Merging Sort	(drum or I.A.S.)
Extraction Sort	(I.A.S.)
Exchanging Sort	(I.A.S.)

Subroutines are also available which facilitate the insertion of a record into its appropriate place within a string of variable-length or fixed-length records.

Merging Sort

Sorting by merging consists of taking two or more ordered groups of data (called strings) and collating these into one ordered group or string. When data are presented in random order, straight merging may be employed. That is, the initial strings are considered as being composed of one item only. The routines available for I.A.S. sorting are all able to employ the straight merge because of the fast internal speed. The method is as follows: The first two keys are compared and their corresponding items placed in their correct order in one receiving area. The next two items are merged in a similar manner and the result placed in a second receiving area. Each new string subsequently created from a pair of original items is then placed alternately into one of the two receiving areas, after the strings already contained in the area. When all the data have been examined in this way the first pass is complete and two streams of data now exist, each consisting of ordered strings of pairs of the original items. The two receiving areas are now used as source areas and the process repeated using two other receiving areas. This time the pairs are merged into groups of four. The data are thus passed back and forth, the areas alternating as receiving and source areas until one completely ordered string emerges. The sort is then complete. This method of sorting by merging using two input areas and two output areas is sometimes called a two-on-two sort.

EXAMPLE OF A 2 ON 2 STRAIGHT MERGE

Initial Order	1st Pass		2nd Pass	
	Area 1	Area 2	Area 3	Area 4
8	4	1	1	2
4	8	7	4	3
7	2	3	7	5
1	9	5	8	9
9	6		6	
2				
3				
5				
6				

3rd Pass		4th Pass	
Area 1	Area 2	Area 3	Area 4
1	6	1	
2		2	
3		3	
4		4	
5		5	
7		6	
8		7	
9		8	
		9	

In practice, areas 1 and 2 are combined into a single area the length of the original data area, and areas 3 and 4 are combined to overwrite the original data area.

For drum sorting, records are stored in blocks. These blocks are of a given decade length, usually 20 decades to make channel transfers possible although 10 decades are often used on 400 I.A.S. machines.

The first part of a drum sort entails merging the records within each block using an I.A.S. merging sort and then storing these blocks on the drum.

A drum merging sort is then used to sort these blocks into their own area on the drum in a sequence defined either by an index in I.A.S. or by a word in each block containing the address of the next block. To do this pairs of blocks are brought into I.A.S., merged together and returned to the drum, giving strings of pairs of ordered blocks. These pairs of blocks are then merged to give strings of four blocks in order. This process is repeated until there is only one string. A Drum Reshuffle routine uses the index to put the blocks thus ordered into a completely sorted file on the drum.

Extraction Sort

An extraction sort entails the sorting of records according to a keyword which is the first word of a record. Routines are available for an extraction sort on both variable-length and fixed-length records.

Basically, an extraction sort involves the extraction of the keywords and their addresses from the original date for an index which is in effect a set of two word records. These key records are sorted by a merging sort, and the addresses are then used to transfer the original records from their data areas to the output area in ascending keyword order. One advantage of using this method is that the I.C.T. subroutines allow the output area to be used as a working area to the index merging sort so that no extra storage area is required other than the original data and output area storage.

Exchanging Sort

An exchanging sort entails the sorting of records (according to a keyword which is the first word of a record) within the original data storage area. The method given below is used in the standard I.C.T routine.

The first key is compared with one half-way through the list of data. An exchange takes place if necessary. The second two keys of each half are compared and so on until all the data has been covered. A second pass over the data is made, the distance between the two keys being compared now being half what it was before. Passes continue, the distance between keys being halved at each successive pass, until finally each key is compared with the next. The sort is then complete. If at any time during a pass an exchange takes place, the lesser item is further compared with earlier items and shifted to its correct position. If the calculation of the distance between keys to be compared does not produce an integer, the next lowest integer is taken as the value.

COMMERCIAL ROUTINES 5.8

The I.C.T. Subroutine Library contains a number of subroutines which cover some of the essential commercial programs that are commonly required, e.g. P.A.Y.E., sterling and decimal conversions, Graduated Pensions etc.

P.A.Y.E. 5.8.1

Routines are available for both monthly and weekly P.A.Y.E. calculations and these routines are constantly amended to conform to Budget changes.

Computation of weekly or monthly P.A.Y.E. contributions is made on data usually required in P.A.Y.E. accountancy, i.e.

Gross Pay brought forward
Gross Pay for period
Tax brought forward
P.A.Y.E. Code Number
Week (Month) Number
Tax Basis Code
Fixed Amount of Tax.

Given the data above, it is possible for the subroutines to compute:

Gross Pay carried forward
Tax carried forward
Tax for period (positive or negative, negative being an indication of a refund of tax).

Graduated Pension Contributions 5.8.2

I.C.T. subroutines are available which compute contributions for the Graduated Pension Scheme for a pay period of one week, multiples of one week or a calendar month. Once the gross pay for the pay period has been determined (i e. gross pay for one month or N weeks etc.) the Graduated Pension Contributions subroutine is entered to calculate the appropriate contributions.

Sterling and Decimal Conversions 5.8.3

A number of subroutines are available which enable such conversions as:

Sterling to £ and decimals of a £
Sterling to pence
Pence to sterling
Sterling amounts (in numeric form) to English (in alphabetic form).

MATHEMATICAL AND STATISTICAL ROUTINES 5.9

Floating-point Arithmetic 5.9.1

In arithmetic calculation it is often convenient to represent numbers by two factors, a decimal quantity and some power of a chosen radix. For example, in denary arithmetic the radix is 10. The number 83256817000000 can be represented by $0.83256817 \times 10^{14}$. Similarly for very small numbers: $0.0000000078325763 = 0.78325763 \times 10^{-8}$. This is known as floating-point representation and can be very useful in overcoming the limiting size of registers. Packages are available for performing the normal arithmetic operations of addition, subtraction, multiplication and division with numbers which range in magnitude from 1×10^{-51} to $1 \times 10^{+49}$ retaining 9 significant digits throughout the range.

Matrix Arithmetic 5.9.2

Many calculations, especially in engineering, are concerned with linear relationships of variables (e.g. Hooke's Law), and problems often arise in the form of several linear equations relating one set of variables $y_1, y_2, y_3, \ldots\ldots y_n$ to another set $x_1, x_2, x_3, \ldots\ldots\ldots x_n$.

This gives n simultaneous equations which written in full are:

$$\begin{array}{llll}
y_1 = a_{11}\, x_1 + a_{12}\, x_2 + a_{13}\, x_3 \cdots\cdots\cdots\cdots & + a_{1n}\, x_n. \\
y_2 = a_{21}\, x_1 + a_{22}\, x_2 + a_{23}\, x_3 \cdots\cdots\cdots\cdots & + a_{2n}\, x_n. \\
\vdots \quad\;\; \vdots & \quad\;\vdots \\
y_n = a_{n1}\, x_1 + a_{n2}\, x_2 + a_{n3}\, x_3 \cdots\cdots\cdots\cdots & + a_{nn}\, x_n.
\end{array}$$

where a_{n1} is the coefficient of x_1 in the expression for y_n. The convenient abbreviated form of this set of coefficients is called a matrix and is written as follows:

$$\begin{array}{llll}
a_{11} & a_{12} & a_{13} \cdots\cdots\cdots\cdots & a_{1n} \\
a_{21} & a_{22} & a_{23} \cdots\cdots\cdots\cdots & a_{2n} \\
\vdots & & & \vdots \\
a_{n1} & a_{n2} & a_{n3} \cdots\cdots\cdots\cdots & a_{nn}
\end{array}$$

The equations are denoted by the single expression:

$y = Ax$

where A is the matrix above of n rows and n columns. A set of subroutines is available for carrying out the common operations of linear algebra, i.e. addition, subtraction, multiplication, transposition, inversion, calculation of eigen roots and eigen vectors, and the solution of simultaneous equations. Facilities are also provided for reading in and printing out a matrix.

Double - length Arithmetic 5.9.3

For use in those cases where the numbers used can vary moderately in size, and in which a high degree of precision is required, a double-length arithmetic package is available which gives facilities for carrying out the normal arithmetic operations with numbers which are 22 digits long instead of the usual 11 digits.

Special Mathematical Routines 5.9.4

Linear Programming is used to optimize a linear function of a set of variables subjected to linear constraints, by means of the Simplex Method. Facilities are provided for taking into account the imposition of upper and lower bounds on the variables.

Statistical Programs are available for the analysis of sets of experimental observations. These have the ability to draw inferences regarding the representation which most closely fits the observations.

Engineering Programs have been developed to evaluate the stresses and deflections induced in common structural forms by the loads imposed upon them.

A set of subroutines are available for the evaluation of commonly occurring Mathematical and Trigonometrical Functions such as exponential, sine, tangent, etc.

Manchester AutoCode 5.9.5

The Manchester AutoCode provides facilities for incorporating into a program MAC routines which have been written independently of that program. Certain general purpose MAC routines are available from the I.C.T. Subroutine Library. The mathematical and statistical library routines therefore fall into three groups:

(a) Complete routines for which the user need only supply data. These may be written either in MAC or in machine code.

(b) Subroutines written in machine code which may be incorporated into machine-coded programs.

(c) General purpose routines written in MAC which may be incorporated into MAC programs.