CCS-E2X4 - Issue 12 - 11th February 2011
Compiled by D. J. Pentecost

SOFTWARE AND SAMPLE PROGRAMS FOR ELLIOTT 401, 402, 403, 405 COMPUTERS

Elliott-NRDC 401

The information in this section consists largely of edited versions of much of the contents of a paper written by D.H. Rees of the Rothamsted Experimental Station (see the references in the final section).

Programming is the art of breaking down a mathematical calculation into a sequence of instructions which the machine can carry out. A few simple examples will illustrate the process.

Example 1. If numbers x, y and z are held in locations 1.00, 1.10 and 1.20, place x – y in 1.30 and x + y – 2z in 1.40.

A0	A2	SFDK	A1	R1	R3	R4	R5
1.127	1.01	6400	1.00	x			
1.01	1.11	6430	1.10	y	x		
1.11	1.13	0440	1.12	0		y	
1.13	1.22	6200	1.20	2z			
1.22	1.24	3450	1.23	x			2z
1.24	1.26	4600	1.25	x-y			
1.26	1.34	3460	1.30	x			
1.34	1.36	4000	1.35	x+y			
1.36	1.38	5600	1.37	x+y-2z			
1.38	—	0600	1.40				

Notes 1. Since R1 is clear before 1.13 is obeyed, the order in 1.13 which gives 2 left shifts produces $0 \times 2^2 + z \times 2^1$.

2. An order such as 1.13 must be programmed first, since both its A1 and A2 are determinate.

3. Some time is saved by temporarily storing the results in the registers. In a large program, these might be occupied by other values, in which case a different method might be desirable.

4. An alternative program, taking more time but fewer orders is as follows:

				R1	R3	R4	R5
1.01	1.22	6210	1.20	2z			
1.22	1.02	6440	1.00	x		2z	
1.02	1.11	6630	1.10	x-y	x		
1.11	1.31	3460	1.30	x			
1.31	1.33	4600	1.32	x-2z			
1.33	1.12	6000	1.10	x+y-2z			
1.12	—	0060	1.40				

Example 2. Form a table of $n \times 2^{-31}$ in 1.00, 1.01, …
$n^2 \times 2^{-31}$ in 3.00, 3.01, …
n = 0, 1, 2, …

				R1	R3	R4	R5
5.00	5.02	0400	5.01	0			
5.02	5.04	0040	5.03			0	
5.04	5.11	0050	5.05				0
→5.11	5.01	5465	3.00	n		n^2	n
5.01	5.03	1660	1.00	n+1			
5.03	5.05	5450	5.04	n			n+1
5.05	5.07	5000	5.06	2n+1			
5.07	5.09	4000	5.08	$(n+1)^2$			
5.09	5.11	0045	5.10			$(n+1)^2$	

Notes 1. The two write to store orders in 5.11 and 5.01 are R5-modified. Each time the loop of orders is obeyed, the contents of R5 are added to these orders before they are obeyed, so that the A1's become successively 1.00, 1.01, 1.02, … It is important to realize that the orders as stored on the drum are unchanged by this process.

2. The program must be stopped before n reaches 257, otherwise overwriting will occur in 5.00 and 3.00.

Nearly all programs are built up of loops of orders as illustrated in example 2. The B-modification feature enables different numbers to be picked up from or written to the store on each passage round the loop. It is clearly necessary for the machine to leave the loop sooner or later and to proceed to the next part of the program. This is achieved by means of a test order, which will lead the machine to one of two possible subsequent orders according to the result of the test. Frequently the number of times the loop has to be traversed will be known and a count is kept. Testing the count enables the machine to leave the loop after the correct number of passages and the count can also be used for B-modification.

<u>Example 3.</u> A set of numbers is held in locations 2.30, 2.31, …, 2.86. Place their sum in 2.00. It is assumed that the sum is within the capacity of the machine.

```
   1.64   1.66   0400   1.65
   1.66   1.68   6430   1.67
┌→1.68   1.70   1002   1.69 ─┐
│  1.70   1.72   0044   1.71  │
│  1.72   1.73   6400   2.30  │
│  1.73   1.75   3000   1.74  │
└─ 1.75   1.68   4430   1.76  │
   1.69   1.71   3400   1.70←─┘
   1.71    —     0060   2.00
   1.67   0.00   0000   0.57 ‖
```

<u>Notes</u> 1. The sum is accumulated in R3, which must be cleared at the beginning of the program.

2. The count number, 57×2^{-31}, is put into 1.67 in the form of an order (‖ indicates a pseudo-order). This avoids the use of a special input routine.

3. The value of the count which is tested is that of the cycle just completed. Hence when the count reaches zero, the number in 2.30 has been added to the sum and the total is complete. Counting is normally carried out from the top downwards as in this example.

<u>Example 4.</u> Numbers x and y are held in locations 1.00, 1.01. Place their product, rounded to 31 binary places, in 1.02.

```
1.127   1.04   6400   1.00        Pick up x
1.04    1.06   6436   1.01        x to R3, y to R2
1.06    1.38   0110   1.07        Form xy
1.38     —     0060   1.02        and store it
```

<u>Notes</u> 1. The parity of the addresses (A0; A2, A1) of the double length order in 1.04 may be written even; even, odd. The rules given in the section on Instruction Sets and Instruction Times then show that

a. the contents of R1 and R2 are not interchanged
b. the contents of R2 are replaced by the number from the store (S = 6, F = 4); the contents of R1 (x) go to the destination (R3).

2. For the multiplication order, A2 – A1 = 31.

(A0 refers to the address of the order in the store, and A2 and A1 are the address fields in the order itself.)

<u>Example 5.</u> Small integers x and y are held in 1.00, 1.01 in the form $x \times 2^{-31}$, $y \times 2^{-31}$. Place $xy \times 2^{-31}$ in 1.12.

```
1.127   1.02   6400   1.00
1.02    1.04   6436   1.01
1.04    1.06   0400   1.05
1.06    1.39   0100   1.08
1.39    1.46   0306   1.41
1.46     —     0600   1.12
```

Notes 1. R1 is cleared before the unrounded multiplication.

2. The product is shifted 3 places to the right to remove the 3 redundant digits at the least significant end of R2, and is left in R1. This takes 2 x 3 – 1 word times. A0 and A1 of the shift order are odd.

Example 6. Replace the contents of 1.00 by its modulus.

```
    1.127  1.01   6400   1.00
┌── 1.01   1.04   0031   1.02
│   1.02   1.04   3610   1.03
└──→1.04    —     0060   1.00
```

Example 7. The digits of a 9-figure integer are punched on successive rows of the tape, starting with the most significant; place this integer x 2^{-31} in 1.00.

```
    1.01   1.03   2400   1.02
    1.03   1.05   6430   1.04
┌──→1.05   1.07   1002   1.06 ──┐
│   1.07   1.09   3440   1.08   │
│   1.09   1.11   0230   1.10   │
│   1.11   1.14   3200   1.12   │
│   1.14   1.16   2000   1.15   │
└── 1.16   1.05   4430   1.17   │
    1.06   1.08   3400   1.07←──┘
    1.08    —     0060   1.00
    1.04   0.00   0000   0.08  ||
```

Notes 1. Multiplication by 10 is achieved quickly and simply by the two orders in 1.09 and 1.11.

Programming: Special Devices.

In this section a few tricks of programming are described which are useful in saving space or time.

1. The result of an output order depends on only the last 5 digits of R1. This means that it is seldom necessary to carry special constants to produce CRLF's or spaces – any order whose A1 ends in the correct 5 digits can be picked up and sent to output. Such orders should be marked on the programming sheet to avoid their being altered by later corrections to the program.

Originally the input and output codes were not the same: since the only output was to the typewriter, this did not matter. (The input code was unchecked binary coding straight into R1 and the output code a 2-out-of-5 code for digits 0 - 9 + – **.** sp CR LF.) This was changed in 1956 to a common code for input and output, with checked and unchecked forms.

To move the paper up 3 lines requires 3 CR LF's and 2 spaces. This may be done in three orders (+ one store location) by storing the following constant –

```
0.000000      01000  11110  01000  11110  01000
              CR LF    sp   CR LF    sp   CR LF
```

in (say) 1.01 and using a loop such as the following –

```
    1.00   1.02   6400   1.01
┌──→1.02   1.08   0320   1.03
└── 1.08   1.02   0002    —
```

This will save appreciable space if the spacing is required several times in one program.

2. A simple way of traversing a loop twice is to set 0 ≠ 1's (note: here ≠ means not equivalent) the first time round and the result (1's) ≠ 1's, producing 0's, the second time and testing for zero. This has the merit of being self-resetting, provided that the 0 is set initially by the program.

3. All programs should be made self-initializing, the necessary orders being included in the program, not set by pseudo-orders on the program tape; This applies to counts, etc. and any orders that may be changed by the program. The program can then be restarted after an untoward incident without feeding in the whole program tape again.

4. Using the initial orders on Track 0 of the store, facilities for reading orders from tape and storing them are always readily available. The easiest way of inserting a few simple constants is therefore to present them in the form of orders. Examples of this have been given above in examples 3 and 7. To convert a positive decimal fraction to a pseudo-order, multiply the fraction successively by 4, 128, 8, 8, 8, 8, 8, 128, each time recording and clearing the integral part. An analogous division process is used to convert integers.

5. When coding a problem, the SFDK's of the order should be written first, with arrows to show the path of the computation and information regarding quantities sent to or from the store, amount of shifts, etc. The addresses can be filled in later as a separate operation. During this second stage, most attention should be paid to the innermost loops of orders, which will encountered the largest number of times. These should be programmed first.

Programming: Track 0.

Track 0 differs from the other tracks in having no writing facilities. The information on it is thus permanent and cannot be inadvertently destroyed by over-writing. The most important part of the information is the Initial Orders. These make up a program that reads in a tape punched with orders and places these orders in the correct positions in the store. This routine has a checking facility: as each order is read in to the store, it is read back and checked against the version which has been constructed from the tape. If desired, the writing can be inhibited by setting the A2 of the hand switches to 0.01, in which case the machine will check the input tape against the pre-existing contents of the store.

The remainder of Track 0 contains certain frequently used subroutines which are described below.

A subroutine is a standard routine that may be used in many different programs or perhaps several times in different parts of the same program. It must therefore be able to direct control to different places each time it is used. This is achieved by providing it with a link order, which is the first order to be obeyed after the orders of the subroutine have been completed. This link order is placed by the subroutine in a certain location and control is finally directed to this location. Information about picking up the link order is always given in the specification of a subroutine.

The Routines of Track 0.

There are four programs permanently available on Track 0, of which two are organized as subroutines.

1. Initial Orders and Post-mortem routine. Start at 0.00 or (with R1 = 0) at 0.22.

2. Division. Enter at 0.112, with x in R3, y in R4, link in R1.
Forms x/y in R1.

3. Print fraction. Enter at 0.52 or 0.53. Prints the fraction x to n digits, rounded off, where x is in R3, n x 2^{-31} in R4, and link in R1. Negative numbers are preceded by – sign, positive numbers by a space or CR LF. Entry at 0.52 causes the number to be preceded by a space; entry at 0.53 causes the number to be preceded by a CR LF.

4. Insert or remove Optional Stops. Start at 0.01, with the A0 of the appropriate order set as A1 on the hand switches. The routine sets the K-digit of the order ≠ 7 (not-equivalent) and returns control to 0.01, which is a stop order, so that the hand switches can be changed in order to modify another order if required.

Elliott 402

There is an example of a 402 machine code program, for calculating a square root, set out in Note 8 (pages 37 to 39) of the 402 programming manual which is reproduced in the section on Instruction Sets and Instruction Times. No other examples of application machine coding have come to light. But a version of the Initial Orders coding is shown as photographs of an extract from a 402 programming manual; this program is on the last ten pages of the 402 part of the section on Instruction Sets and Instruction Times.

However, a two page note has been found in the London Science Museum, under reference 681.32.ELL 402, received 26th June 1978, produced by the Scientific Computing Division of Elliott Brothers (London) Limited, dated 18th March 1958, under reference L134.. This note was produced for the 1958 Physical Society Exhibition. The version below has been carefully re-typed from a scanned photocopy.

It shows examples of the kind of work for which the 402F was being sold to undertake, and although no machine code is listed, there is a fragment of Autocode illustrated, on the second page.

DJP 8th Jan 2011

1958 PHYSICAL SOCIETY EXHIBITION

Demonstration of the

ELLIOTT 402F (FLOATING POINT) DIGITAL COMPUTER

This machine, the latest extension of the well-known Elliott "400" series, consists of a 402E (fixed point) computer with three extra cabinets, and can work under program control in either fixed or floating-point modes.

The machine is demonstrated in four typical application:

(i) Solution of n simultaneous equations ax = b with n right-hand sides, using minimum storage space

(ii) A linear programming problem

(iii) Multiple regression analysis

(iv) Simple autocode programming

Solution of n simultaneous equations ax = b with n right-hand sides, using minimum storage space

The elements of the combined matrix (a,b) are read in, a row at a time; when the (r + 1)th row is read the rows 1,2, ... r have been reduced to the form $M_r \equiv (d_r, a_r, b_r)$ where d_r is a diagonal r x r submatrix and a_r, b_r are submatrices with r rows and n-m, m columns respectively. By subtracting suitable multiples of rows, the matrix (M_r + 1) is then built up. If zero elements are not stored the maximum number of locations required to store M_1 is $\frac{1}{2} n(\frac{1}{2} n + m + 1) \neq \frac{1}{4} n^2$ which occurs when $r = \frac{1}{2} n$.

Without using floating point working at least $\frac{1}{2} n^2$ locations are required for storage. The program consists of approximately 120 orders excluding input and output subroutines.

Linear Programming

Linear Programming is a method of maximising a linear function of several variables which are subjected to a number of constraints expressed as a set of linear inequalities.

Example

Maximise $f = 4x_1 - 3x_2 + 40x_3 + 10x_4$

where

$$x_1 + 11x_3 + x_4 \leq 37$$

$$x_2 + 3x_3 = 29$$

$$x_3 + x_4 \leq 9$$

Solution

$$f = 115$$

$$x_1 = 28, \quad x_2 = 29, \quad x_3 = 0, \quad x_4 = 9$$

The routines being demonstrated use the Simplex Method and can deal with 61 equations with 61 non-basic variables.

/Over

Multiple Regression

Given a matrix of variances and covariances, the program will produce regression coefficients of any selected variate or any other selected variates, together with sums of squares and degrees of freedom required for an analysis of variance. At any stage a variate so far included may be removed from the regression equation. The program is controlled entirely from the hand keys.

Similar programs written for a fixed point machine would require checks on capacity at about ten different points, and should capacity be exceeded at any stage of these points it would be necessary to rescale the data or intermediate values and restart the program from the beginning.

Simple Autocode

The Simple Autocode program will read in a tape consisting of "algebraic orders" and produce an equivalent program in machine orders which can then be read into the machine and obeyed.

In order that the present teleprinter codes might be used the following conventions have been adopted:

*	is used for 'mutiply'
/	is used for 'divide'
)	is used for 'greater than'
(	is used for 'less than'

In a long arithmetic order like that marked '1 below, the functions are performed in the order in which they occur, as can be seen from the explanation on the right.

An example is given below of a sequence of orders which might be required:

Specimen Set of Orders

Order		Explanation
I, A		Input the number A
'1 Z = X + A * 352.79/B - 4.6	(1)	Form $Z = \frac{352.79\ (X + A)}{B} - 4.6$
P, Z 0'		Print Z (print routine no: 0)
A = F 5 (A)		Replace A by f_5(A) (where f_5 is probably a library subroutine, e.g. logarithms or sine).
T, A) 0.1 @[†] 1		If A > 0.1 return to line (2)
S,		Otherwise stop.

Scientific Computing Division
ELLIOTT BROTHERS (LONDON) LIMITED
Elstree Way, Borehamwood, Hertfordshire.
ELStree 2040

18.3.58. Printed in England

[†] *(This character is unclear on the copy of the document, and may not be an @ sign)*

Elliott 403 - WREDAC

[Compiled from various sources by D.J.Pentecost]

The program on this and the following six pages appears to be a library program, and was written at the Weapons Research Establishment, Salisbury, South Australia. It was started in November 1961, and published on 15th January 1962. It was Alex Reid's first program, and has been taken from his website, which at the time of writing (April 2006) was at www.general.uwa.edu.au/u/alex/Welcome-5.html. He has kindly given permission for the program to be reproduced here. Because of the difficulty of copying and re-publishing what appears to be a scanned photocopy on Mr. Reid's website, much of the program specification has been carefully re-typed and checked, and is reproduced below in its original format.

****

UNCLASSIFIED

WEAPONS RESEARCH ESTABLISHMENT
MATHEMATICAL SERVICES GROUP

WREDAC PROGRAMME SPECIFICATION

A

REFERENCE : G 11.1.1

TITLE : Transposition of signs on paper tapes

AUTHOR : T.A.Reid **VETTED : B.Biggins**

DATE : 15/1/62 **APPROVED : P.N.L.GODDARD**

PURPOSE : In the three columns of numbers output on paper tape from Boscar film readers the second and third are followed by signs. This programme transposes the signs to the front of the numbers so that the tapes can be used on the Libroscope plotter. The telereader is able to punch the sign either before or after the number, but it is customarily placed after the number, for use in IG.1.22.

RESULTS : Output is of the same form as the input data, i.e. it consists of three columns of four figure numbers, the second and third columns having signs in front of them. (See B)

WR 18673

REFERENCE : G 11.1.1

B

DATA REQUIREMENTS : The first character of the data tape may or may not be a β; data should then be punched in the sequence :

....xxxxβxxxx±βxxxx±βγβ....

or ....xxxxβxxxx±βxxxx±βγ....

or ....xxxxβxxxx±βxxxx±γβ....

or ....xxxxβxxxx±βxxxx±γ....

and should end with γz or γβz.

C

METHOD : (i) Input and output of blank tape at the beginning of the data tape is carried out until the first non-blank character is read, when it is punched. If this is β the next character will also be punched.

(ii) The next three digits and the following β are input and punched.

(iii) The following four digits and sign are input and stored.

(iv) The sign is punched

(v) The four digits in storage are punched

(vi) The next character is input and tested : if the data sequence has been upset and this is not β a dynamic stop occurs. If it is β, it is punched.

(vii) The next four digits and sign are input and stored.

(viii) The sign is punched.

(ix) The four digits in storage are punched.

(x) The following β's and γ are input and punched.

STORAGE DISTRIBUTION :

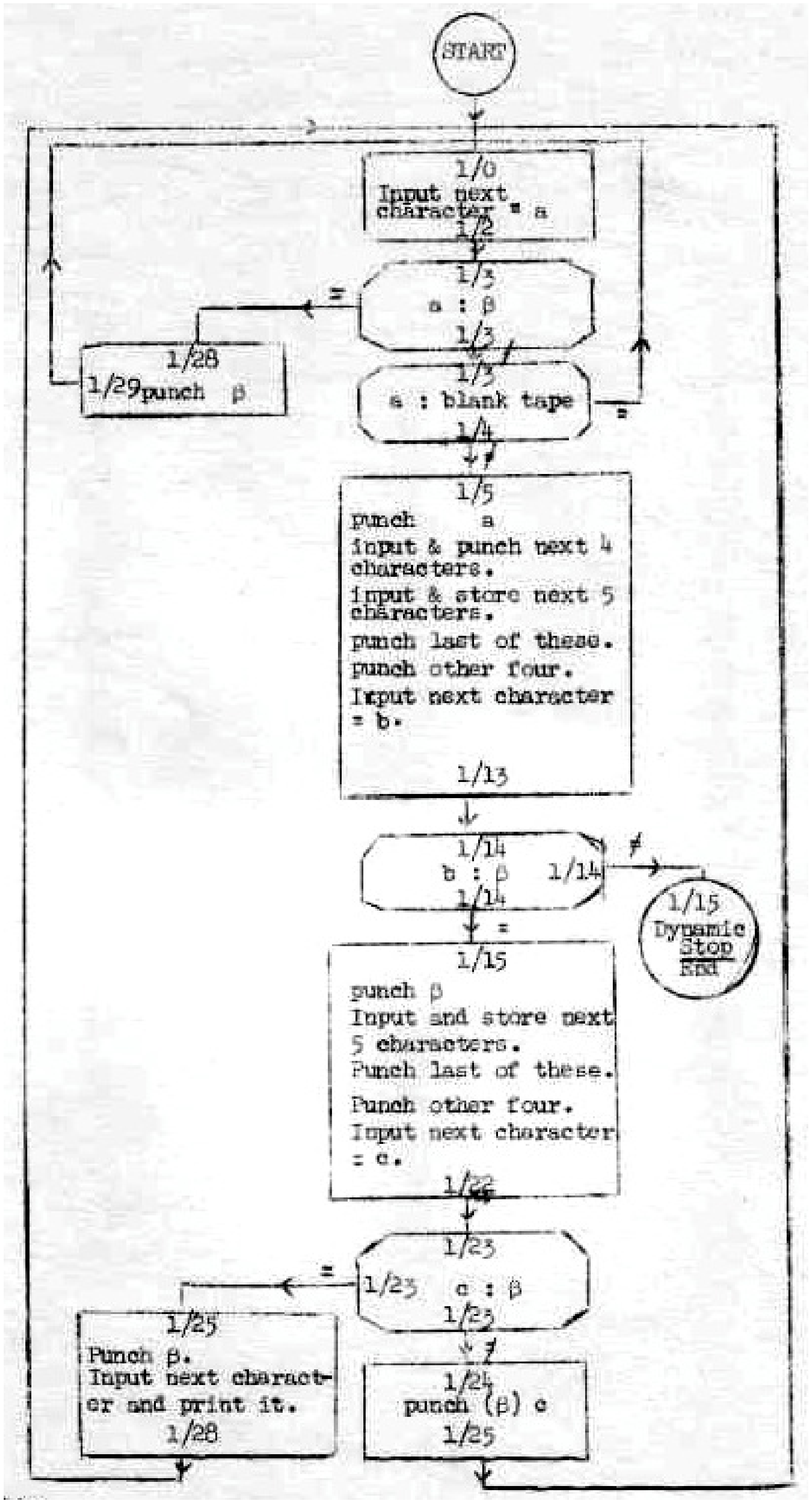
(i) <u>B-lines used</u> : B-line 1 is used.

(ii) <u>High Speed Store</u>

from	1/0 to 1/30	Master routine
from	1/60 to 2/0	Working locations

DATA STORAGE DETAILS : Data is not stored but is output immediately it has been input.

FLOW CHART. REFERENCE : G 11.1.1



REFERENCE : G 11.1.1

PROGRAMME TAPE ASSEMBLY :

QA 64 Φ

$\boxed{\text{Master routine}}$

QC Φ | GE 64 α

D

INPUT/OUTPUT EQUIPMENT :
1 paper tape reader
1 paper tape punch

OPERATING TIME : 44 characters and read and output per second i.e. 138 lines are processed a minute.

OPERATING INSTRUCTIONS :

(i) Setting up : none

(ii) Programme and Data Feed:
Read in programme to
OPSTOP 1 : Insert data tape and single shot.
Results are output as fast as data is input.

COMPUTATION ENDS ON GE 1/15 α

RESET TO OPSTOP 1 by obeying: GE 1/1 α.

(iii) Output :
The output tape is of exactly the same form as the data tape but with the two signs transposed into the positions as shown:

....xxxxβ±xxxxβ±xxxxβγβ....

(iv) Programmed Halts :

Order Register	Sequence Control	Cause and remedy
OPTIONAL STOPS :		
CH α	1/0	OPSTOP 1 as above
DYNAMIC STOPS :		
GE 1/15 α	1/15	(a) if end of data tape : end of computation. (b) if not end of data tape : sequence of data characters has been upset: move data tape to next γ and obey GE 1/22 α.

UNCLASSIFIED
WRE DIGITAL COMPUTER
PROGRAMME SHEET

REFERENCE : G 11.1.1

TITLE : Transposition of signs on paper tape

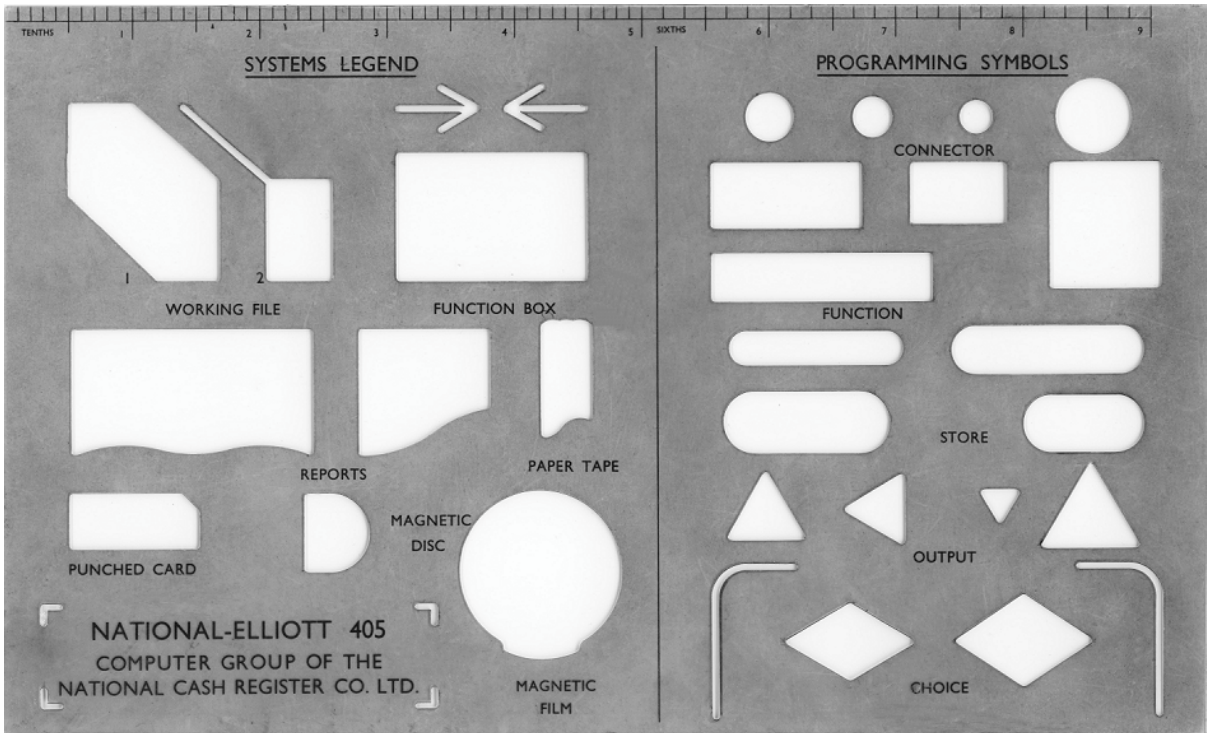
QA 64 α

	NO.	ORDERS (EVEN)			ORDERS (ODD)				NOTES
4,25, 28,29	0	CH		α	ST		α		
→	1	ST		α	CL		α		
	2	IM		∅	SA	30	Aα		tests for 1st characters
	3	CH	28	Aα	AD	30	Aα		prints β if there + 1st character
	4	OM		∅	CH	1	Aα		
	5	SB	509	α	CL		α	← 7	outputs next 3 digits + β.
	6	IM		∅	OM		∅		
	7	JO	5	Aα	SB	508	α		
	8	CL		α	IM		∅	← 9	124 = 1st digit
	9	CL	128	β	JO	8	Aα		125 = 2nd "
									125 = 3rd "
									127 = 4th "
									128 = sign "
	10	CA	128	α	OM		∅		outputs sign
	11	SB	509	α	CA	127	β	← 12	
	12	OM		∅	JO	11	Aα		outputs 4 digits
	13	CL		α	IM		∅		tests for and outputs β
	14	SA	30	Aα	CH	15	Aα		
15 →	15	GE	15	Aα	AD	30	Aα	← 14	
	16	OM		∅	SB	508	α		
	17	CL		α	IM		∅	← 18	stores 4 digits + sign
	18	CL	128	β	JO	17	Aα		
	19	CA	128	α	OM		∅		outputs sign
	20	SB	509	α	CA	127	β	← 21	
	21	CM		∅	JO	20	Aα		outputs 4 digits
	22	CL		α	IM		∅		
	23	SA	30	Aα	CH	25	Aα		tests for β
	24	AD	30	Aα	OM		∅		outputs next character, γ.
	25	GE	1	Aα	AD	30	Aα	← 23	
	26	OM		∅	CL		α		outputs β γ.
	27	IM		∅	OM		∅		
	28	GE	1	Aα	AD	30	Aα	← 3	
	29	OM		∅	GE	1	Aα		outputs β
	30	CH		α	AO		α		

WR 18673

National-Elliott 405
[Compiled by D.J.Pentecost]

Below is a copy of the standard 405 program flowcharting template in use in the early 1960s.



1. Automatic Timing Routine

An important consideration for computers of this generation was to optimise the coding of critical parts of programs. For example, batch processes, such as large payrolls and stock control, involving data held on several reels of magnetic tape, could take several hours to run. So an hour saved by efficient coding was a bonus well worth achieving, for computer time was often at a premium, with jobs being tightly scheduled and competing with each other for access to the computer.

Another consideration was the fact that computer breakdowns occurred not infrequently, so if the time taken to run a job could be minimised, there was a greater chance that it could be completed before the next breakdown occurred.

So programmers sometimes needed a tool to measure how long a critical part of a program was taking to run, in order if necessary to re-code it to reduce run time. Elliotts provided for this purpose a library program, known as the Automatic Timing Routine, or 405 PT 02.

This timing routine is set out below as a re-typed copy of an original specification document followed by photo-copies of the two original program sheets, (both now in the London Science Museum), a printout resulting from passing the paper tape of the program through a tape reader and printer, and for the sake of completeness, to show all the principal forms of a program, a copy of the unrolled reel of paper tape holding the program.

Apart from its academic interest, the program illustrates the use of the following 0-code orders: 0, 1, 2, 3, 4, 6, 8, 9, 10 and 11.

AUT0MATIC TIMING ROUTINE

1. DESCRIPTION

To measure in word times the time taken by any program which does not obey an output compiler order; the program may take a time ranging from milliseconds to hours.

2. DEFINITION

The program being timed is referred to as P.

3. COMPUTER REQUIREMENTS

6-bit output line assembler
Immediate access locations 16 to 31

4. LOCATIONS USED

I.a.s. 1, 193 to 253 and 511.

5. VARIANT ROUTINES

If the computer has a 5-bit output line assembler, the routine must be amended thus: 217) 113'0/7,5'480.

If locations 16 to 31 are not i.a.s.'s change the order pair in 247 to 9'228,2'253.

6. METHOD OF USE

The program consists of two subroutines, which "start the clock" and "Tell the time". (Hereafter referred to as S-R(i) and S-R(ii) respectively).

The procedure is for the user to estimate the maximum possible time taken by P, enter S-R(i), obey P and finally enter S-R(ii).

In order to function, S-R(ii) requires three values which are placed in it by S-R(i). Therefore if P uses locations 193 to 253, the routine must be stored away, either elsewhere in the working store or on the disc, after obeying S-R(i), and of course be restored to 193 to 253 before entering S-R(ii). It should be noted that the instructions required to achieve this occur after S-R(i), and before S-R(ii), and are taken by the routine to be a part of P; thus their time will be included in the result.

N.B. Before reading in the tape, the sector to contain the routine must be in location 400 at scale 2^{-24} (collated).

(a) S-R(i)

Parameter: Before entering S-R(i), the user must place in location 195 an estimated maximum time taken by P (in word times) at scale 2^{-31} (T_{max}). This value, once placed, remains in 195, and may therefore be used if required for timing several programs. T_{max} must be > T, where T is the exact time of P in word times.

Entry is into location 193:

```
  x) 171'0,8'193
x+1) f'n, ....
```

f'n, is the first instruction of the program being timed.

Exit is normal; ,8'0/1.

(b) S-R(ii)

Entry is into location 194:

```
y-1) ..... ,F'N
  y) 171'0,8'194
```

,F'N is the last instruction of P.

Exit is normal: ,8'0/1.

T, the time measured, is in both the accumulator and location 196 at scale 2^{-31} on exit.

7. THE TIME MEASURED

```
  x) 171'0,8'193      Enter S-R(i)
x+1) f'n, . .       ⎫
. . . . . . . . .   ⎬ Program to be timed
y-1) . . . ,F'N     ⎭
  y} 171'0,8'194      Enter S-R(ii)
```

The time measured is that from coincidence with x+1) until coincidence with y).

Care must be taken when assessing the time measured, if P contains film, disc or input orders. (Not output: see note 10(a)). Suppose that f'n, is a disc order. If shortly before entering S-R(i), another disc order had been obeyed, and there had been no following delaying order referring to this disc order, although the time measured is still that from x+1) to y), this time includes that due to the delaying action of the first disc order on f'n, the second disc order. A similar effect occurs if ,F'N is a 120' disc order; it is clear that the total time for this order will not be measured; but a 121' order (which would necessarily read S-R(ii) to the working store) will be measured.

The range of time measurable is from zero to approximately 2^{32} word times.

Provided that location y) is not an i.a.s., the time measured will be correct; but if it is an i.a.s., the time measured may not be correct, since the routine assumes that $|y)|_{16}$ is the true word time of y), which is not necessarily so. In this case the true time of P is given by the expression

$$T + C_2 - 8 \times \frac{|C_2|}{C_2} - K + 8 \times \frac{|K|}{K}$$

where T = time of P given by the computer

$C_2 = |13 \times \text{link of S-R(ii)}|_{16} - 10$

$K = |13 \times [\text{true word time of y)} + 1]|_{16} - 10$

and for any quantity Q, when Q = 0, $\frac{|Q|}{Q} = 1$.

8. **TIME TAKEN BY S-R(i)**

```
  x} 171'0,8'193
x+1) f'n, . . .
```

The time taken is that from coincidence with x) until coincidence with x+1). 66 word times is the time taken, excluding the times for coincidence of the entry and exit orders, 8'193 and ,8'0/1. Clearly the total time depends upon the times for these two orders, which in turn depend upon whether locations x) and x+1) are i.a.s.'s. If they are not i.a.s.'s, the time for the two ,8' orders is invariably 17 word times, making the total time 83 word times.

9. **TIME TAKEN BY S-R(ii)**

```
y-1}  . . . ,F'N
  y) 171'0,8'194
y+1)
```

The time taken is that from coincidence with y) until coincidence with y+1)

(a) Provided that T_{max} is not too small, the time taken in word times is:-

$$T_{max} - |T_{max}|_{16} - T - C_1 - C_2 + 8\frac{|C_1|}{C_1} - 8\frac{|C_2|}{C_2} + 295,$$

if location x+1) is not an i.a.s. If location x+1) is an i.a.s., the time is 48 word times less.

$$C_1 = |13 \times \text{link of S-R(i)}|_{16} - 4$$

These times are exclusive of the times taken for coincidence of the entry and exit orders ,8'194 and ,8'0/1, which invariably total 17 word times if locations y) and y+1) are not i.a.s.'s.

(b) If T_{max} is too small (see note 10(b)), 130 word times is the time taken, exclusive of coincidence times for the entry and exit orders.

10. RESTRICTIONS

(a) The routine cannot be used to time a program which uses the output compiler or the output line assembler.

(b) If T_{max} is too small, T, the result given, is equal to T_{max}. The copy of the result, c(196), also = T_{max}.

(c) Location 511 should preferably not be used. If its use is essential, it must be cleared before entering S-R(ii). If it is not cleared, and T_{max} is not too small, the routine assumes that T_{max} is too small, and note 10(b) applies. If 511 is used and is not cleared before entry to S-R(ii), and further, T_{max} is too small, the computer loops the instructions in locations 201 to 204 indefinitely; should this happen, depress the step key and transfer control to location 207; this sets T = T_{max}, as in note 10(b).

11. CONTENTS OF LOCATIONS AND REGISTERS

	Entry	Exit
S-R(i)	*195: $T_{max}.2^{-31}$	195: $T_{max}.2^{-31}$ 511: zero K-reg: 1.2^{-31}
S-R(ii)	195: $T_{max}.2^{-31}$ 511: zero K-reg: 1.2^{-31}	Acc: $T.2^{-31}$ 196: $T.2^{-31}$ 195: $T_{max}.2^{-31}$ K-reg: 1.2^{-31}

*To be set by the user

Single length multiplication is set by S-R(ii).

12. USEFUL CONSTANTS AVAILABLE

c(222) = c(232) = 1.2^{-31}
c(197) = 0
c(226) = 1.2^{-31}
c(233) = 10.2^{-31}
c(252) = 16.2^{-12}

13. TECHNIQUE USED

Provided that the program to be timed does not use the output compiler or the output line assembler, output via the assembler may take place simultaneously with the program.

Suppose that just prior to obeying the program, output of a number $N.2^{-31}$ is started via the line assembler. If the K register is set at 1.2^{-31}, the time taken for output is $(N+n)$ word times, where n is a small calculable correction. Provided that $(N+n) > T$, where T is the time taken by the program, then the program will be obeyed before outputting is completed.

Let $(N+n) - T = T_1$, i.e. $T = (N+n) - T_1$.

If N is known and T_1 can be measured, it follows that T can be calculated.

Thus the problem of timing the program reduces to that of measuring T_1, the excess time of outputting over the time taken by the program. By setting the combined count appropriately, it can be arranged that the number output will be placed in location 511; so if a loop is entered immediately after the program has been obeyed, which both counts the number of times it is cycled and examines c(511), T_1 may be calculated, for if each cycle takes w word times, and the number of times cycled before 511 is filled is p, then $T_1 = pw$ word times. (In fact pw will be slightly larger than T_1, since the new c(511) will be examined a small interval after it is output, but the difference is calculable, for the exact outputting time is known).

Apart from the calculations and corrections mentioned above, two more adjustments are necessary in order to find T exactly. Clearly, since outputting is started in S-R(i), the time measured depends upon the time taken for exit from this subroutine; it also depends upon the time taken for entry into S-R(ii). These times are accounted for by examination of the two links; the links are also examined to see if i.a.s.'s are involved, in which case further corrections are applied.

14. OPERATING NOTE

Immediately after P has been obeyed, and whilst T_1 is being measured, a high-pitched note is emitted from the loud speaker; note that this is not a dynamic stop, and that a count may be seen to be increasing in both the accumulator and in i.a.s. 1; the note is emitted for approximately the difference in time between T_{max} and the time taken by P.

D.J. Pentecost, 405 PT 02/DJP

NATIONAL-ELLIOTT 405 PROGRAM

SHEET 1 OF 2

TITLE AUTOMATIC TIMING ROUTINE

LIBRARY ROUTINE 405 PT 02/DJP

PROGRAMMER D. J. PENTECOST.

DATE MAY 1961

COST CODE 310/833

DIRECTORY 192/400: @ cr lf bl

LABEL * cr lf bl

BLOCK

SECTOR

PREVIOUS ADDRESSES	Address	F	A	B	F	A	B	NOTES
	192	8	0	3				Not in use
	193	11	1		8	210		ENTRY S-R(i)
	194	11	1		8	200		ENTRY S-R(ii)
	195				≡			Maximum estimated time = T_{max} $(\times 2^{-31})$
	196				≡			Actual time taken = T $(\times 2^{-31})$
	197				$0			
202 →	198	11	1		3	220		
	199	9	245		8	207		Estimate too small
194 →	200	10	227		11	222		Store link of S-R(ii)
204 →	201	10	1		11	511		
	202	3	197		9	198		Measure T_1
	203	11	1		0	225		
	204	8	201					
	205				$32			
	206				≡			$T_{max} - \|T_{max}\|_{16} + 32 = N$, number output
199 →	207	11	195		8	242		
	208				13			
	209				$15			
193 →	210	10	221		11	232		Store link of S-R(i)
	211	113	0	3	4	0		
	212	10	511		8	213		
	213	11	195		8	214		
	214	1	209		8	215		Form $N = T_{max} - \|T_{max}\|_{16} + 32$
	215	1	226		3	205		
	216	0	195		10	206		
	217	113	0	7	4	480		
	218	11	206		8	219		
	219	114	1	3	8	0	1	EXIT S-R(i)
	220				$4			
	221				≡			Link of S-R(i)
	222				$1			
240 →	223	0	225		8	241		

NATIONAL-ELLIOTT 405 PROGRAM SHEET 2 OF 2

TITLE AUTOMATIC TIMING ROUTINE

LIBRARY ROUTINE 405 PT 02/DJP

PROGRAMMER D. J. PENTECOST.

DATE MAY 1961

COST CODE 310/833

DIRECTORY	@ cr lf bl	BLOCK
LABEL	* cr lf bl	SECTOR

PREVIOUS ADDRESSES	Address	F	A	B	F	A	B	NOTES
	224	8	0	3				Not in use
	225				$ 16			
	226				$ −1			
	227				≡			Link of S-R(ii)
246, 248 →	228	11	221		8	229		
	229	172	0		6	234		Form $C_1 + 8 - 8\frac{\lvert C_1\rvert}{C_1}$ to correct
	230	1	209		2	220		for exit time from S-R(i)
	231	9	235		8	237		
	232				$ 1			
	233				$ 10			
	234				3			
231 →	235	0	225		8	237		
244 →	236	172	0		8	0	1	EXIT S-R(ii)
231, 235, 249 →	237	0	1		10	1		
	238	11	227		8	239		Form $C_2 + 8 - 8\frac{\lvert C_2\rvert}{C_2}$ to correct
	239	6	208		1	209		for entry time to S-R(ii)
	240	2	233		9	223		
223 →	241	0	1		3	206		
207 →	242	10	196		11	227		
	243	10	1		11	196		
	244	8	236					
199 →	245	11	221		3	251		
	246	9	228		2	252		
	247	9	249		2	253		$16 \leq x+1) \leq 32$
	248	9	228		8	249		$0 \leq x+1) \leq 3$
247 →	249	4	0		8	237		
	250	8	0	3				Not in use
	251		31					
	252		16					
	253		12					
	254	8	0	3				Not in use
	255	8	0	3				Not in use

```
192/400:@
*405 PT 02/DJP
8'/3,400:
11'1,8'210
11'1,8'200
0
8'/3,400:4
0
11'1,3'220
9'245,8'207
10'227,11'222
10'1,11'511
3'197,9'198
11'1,'225
8'201,
32
8'/3,400:14
11'195,8'242
,13'
15
10'221,11'232
113'/3,4'
10'511,8'213
11'195,8'214
1'209,8'215
1'226,3'205
'195,10'206
113'/7,4'480
11'206,8'219
114'1/3,8'/1
4
8'/3,400:29
1
'225,8'241
8'/3,400:32
16
-1
8'/3,400:35
11'221,8'229
172',6'234
1'209,2'220
9'235,8'237
1
10
,3'
'225,8'237
172',8'/1
'1,10'1
11'227,8'239
6'208,1'209
2'233,9'223
'1,3'206
10'196,11'227
10'1,11'196
8'236,
11'221,3'251
9'228,2'252
9'249,2'253
9'228,8'249
4',8'237
8'/3,400:58
'31,
'16,
'12,
8'/3,400:62
8'/3,400:63
)
```

Above, on the previous two pages, is a copy of the programmer's original program sheet.

To the left of this note is its printed representation after being punched on to paper tape, and below on the following page is a copy of the paper tape itself.

The original program sheet above is functionally accurate, and as far as was known, contained no errors.

When the program was produced on paper tape for distribution to potential users, some additions were made, compared with the hand-written original version, in order to conform to some programming standards which were introduced at a later date.

These were:-

1) To insert the dynamic stop instruction 8'/3, into words to which control should (all being well) never be passed, and

2) Into the right hand side of such words also to place the address of the word containing the dynamic stop instruction, in order to assist the operator to determine where the error had occurred, should the dynamic stop ever be obeyed. (This was achieved by using the device "400:", which added the contents of location 400 to the value following the colon, where location 400 was previously set up to hold the address in working store of the first word of the program, as it was loaded into the computer.

The above accounts for the minor differences between the hand-written version of the program above, and its other two paper tape representations shown here and below.

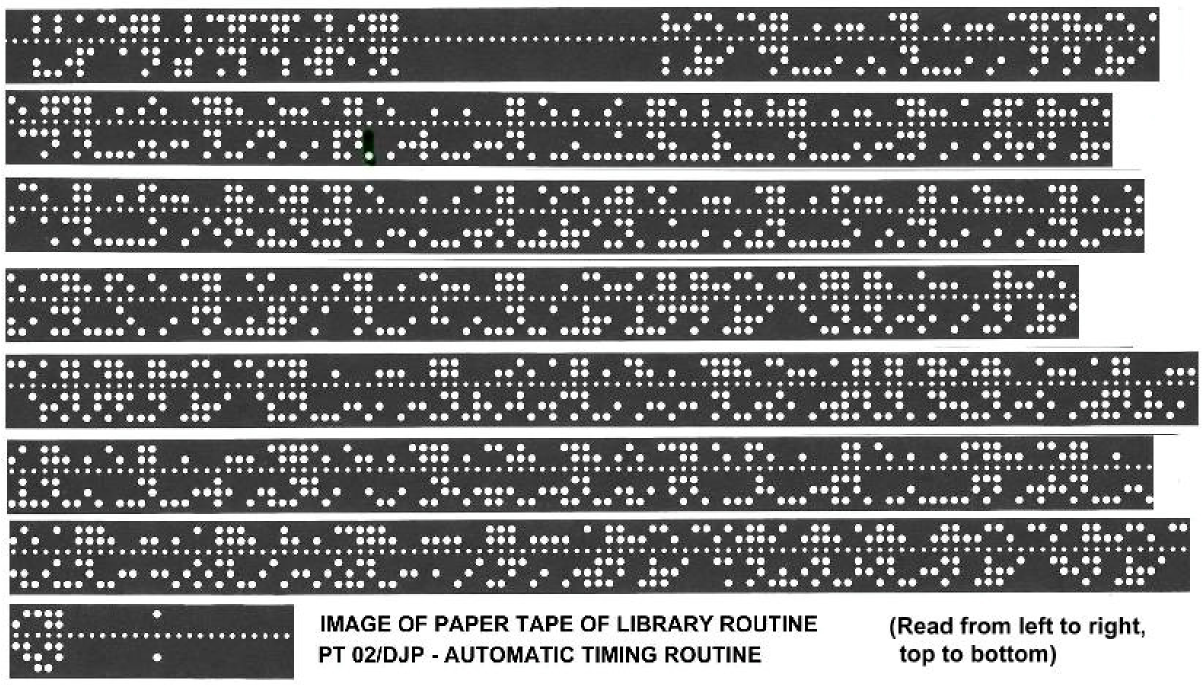


IMAGE OF PAPER TAPE OF LIBRARY ROUTINE PT 02/DJP - AUTOMATIC TIMING ROUTINE

(Read from left to right, top to bottom)

2. Disc Sector 0 routines

On the following two pages is a listing of the two standard paper tape input routines held on sector 0 of the 405 disc, known as FIR (Fast Input Routine), and FGP100, which allowed programs recorded on paper tape to be input to the computer.

(No specification or description has been found)

National-Elliott 405 - Disc Sector 0: FGP100 and Fast Input Routine
Interpreted by H. R. Lawrence, 12th May 2006

```
128/0@
```

Common ENTRY to FGP100 and Fast Input Routine (FIR)

```
129 → 128::  11'164  ,107'  0     Form input character -1 in accumulator
      129::   9'128  ,107'  0     If blank then loop to ignore; else add next input char
      130::   2'184  ,  8'167     Subtract 50.2^(-31);  go to decide if FGP100 entry
```

Directory set-up

```
145 → 131::  11'  3  ,  5' 16     Shift directory address from LS half to MS half
      132::   5'  3  , 10'  2     Shift WS directory address to 2^(-12) and store
143 → 133::  11'159  , 10'  1     Count -5.2^(-12) for selecting appropriate shift
      134::   4'  0  , 10'160     Clear word assembly store
154 → 135::   4'  0  , 10'  3     Clear number assembly store
158,183→136:: 4'  0  ,107'  0     Input next character to accumulator
      137::   2'161  ,  9'139     Subtract 31.2^(-31);  if not 'ls' go to 139
'ls'  138::  11' 0/3 ,  7'511/2   If 'ls' then trigger: Jump to last location assembled
                                  with accumulator holding 11'0/3,7'511/2.  Note the
                                  '7' instruction is actually obeyed  as '8' (jump) and
                                  the address becomes the contents of IAS2 less 1.

137 → 139::   0'162  ,  9'144     Add 2.2^(-31); if  not 'cr' or 'lf' go to 144
'cr'/'lf' 140:: 11'  3  ,  0'160  Add last number assembled to word assembled
      141::  10'  0/2, 11'  2     Store in next store location: Update store location...
      142::   0'163  , 10'  2     ... pointer by adding 1.2^(-12) to IAS2
      143:: 107'  0  ,  8'133     Ignore 'lf' and loop back to start next word

139 → 144::   2'164  ,  9'155     If not 'sp' go to continue assembling current number
'sp'  145::   2'  1  ,  9'131     If  only directory address input so far, go to set it up
      146::  11'160  ,  0'  3     Add assembled number to so far assembled word
      147::   8'153/1,  0'  0     Jump to decode shift-table below

      148::   5'  9  ,  8'154     SHIFT
      149::   5'  3  ,  8'154         JUMP-TABLE
      150::   5'  4  ,  8'154             TO PLACE
      151::   5'  9  ,  8'154                 PARTS OF
      152::   5'  3  ,  8'154                     A WORD
      153::   8'153  ,  0'  0     Dynamic stop: too many spaces this line
```

148-152 entry: Assemble decimal number

```
      154::  10'160  , 12'135
144 → 155::   0'165  ,  1'166     Add 28 and collate 15.2^(-31) to obtain digit
      156::  10'  3  ,  4' 10     Add this to 10 times assembled # so far
      157::   0'  3  , 10'  3     ...and store back in IAS3 thus compiling decimal #
      158::   8'136  ,  0'  0     Loop back for next digit or end of number
```

Constants and workspace

```
      159::  432=496 ,  0'  0     -5.2^(-12)
      160::  (0'  0  ,  0'  0)    Stores word as it is being assembled
      161::   0'  0  ,171'  0     31.2^(-31)  ('ls' in telecode)
      162::   0'  0  ,  0'  0/1   2.2^(-31)
      163::   0'  1  ,  0'  0     1.2^(-12)
      164:: 177'511  ,177'511     -1.2^(-31)
      165::   0'  0  ,  0'  3/2   28.2^(-31)  ('sp' in telecode)
      166::   0'  0  ,170'  0     15.2^(-31)  (collate constant to extract digit value)
```

Select program FGP100 or FIP:

```
130 → 167::  10'  1  , 11'  1     Copy to IAS1 the sum of the first two chars -51.2^(-31).
      168::  12'182  ,103' 11     If first 2 chars not "'fs'@" run FGP100;  else start
                                  FIR here: Input store directory address to IAR.
FIR   169:: 102'  2  , 10'  1     Skip 'cr' & 'lf' and store current directory
                                  location address in IAS1.
      170::  11'  1  , 10'  2     Prime input total in IAS2. with directory address
```

Next line:

```
175 → 171::  102'  1 ,103' 11    Input 1 telecode character to accumulator
                                  and input next location or trigger value to IAR.
      172::    3'185 ,  9'176    Negate & add 6.2^-(31).  If negative then not "*"
                                  so "(" assumed (trigger).
*     173::  102'  2 , 10'  2    Next location value from IAR and skip "'cr' 'lf'"
      174::    0'  2 , 10'  2    Accumulate sum-check total in IAS2.
      175::   10' 0/1, 12'171    Store next location value, increase location address
                                  pointer and loop back for next line unless end of store.
```

Trigger:

```
172 → 176::  102'  3 , 10'  2    Skip "'cr' 'lf' =" and ....
      177::    0'  2 , 10'  2    ...add trigger to accumulated sum-check in IAS2
      178::   10'  3 ,103' 11    Place trigger in IAS3 and input sum-check to IAR
      179::  102'  2 ,  2'  2    Ignore "'cr' 'lf'" and form input sum-check
                                  minus accumulated sum
      180::   10'  1 ,  0'  0    Store sum-check difference in IAS1
      181::   12'186 ,  8'  3    Obey trigger, unless sum-check error
```

Here starts FGP100:

```
168 → 182::    0'191 ,  1'166    Sum of the first two chars -27.2^(-31) masked
      183::   10'  3 ,  8'136    Store first digit of WS directory in IAS3 and loop
```

Two more constants

```
      184::    0'  0 ,  0'  6/1  50.2^(-31)
      185::    0'  0 ,  0'  0/3  6.2^(-31) or telecode "="
```

Sum-check error:

```
181 → 186::  110' 63 ,110' 61    Output "'ls' 'cr'"
      187::  110' 62 ,110' 37    Output "'lf' E"
      188::  110' 50 ,110' 50    Output "R R"
      189::  110' 47 ,110' 50    Output "O R"
      190::  110' 27 ,110'  4    Output "'fs' 4"
      191::    8'191 ,  0'  3    Dynamic stop. Also used in 182 as 24.2^(-31) by
                                  collating out 8'191
```

Meaning of abbreviations used:

Chars: **'fs'** = figure shift; **'ls'** = letter shift; **'sp'** = space;
'cr' = carriage return; **'lf'** = line feed

IAR = input assembly register; **ACC** = accumulator; **IASn** = immediate access register n=1,2,3
t/c = telecode; **LS** = least significant half of word (digits 2^(-16) to 2^(-31))
MS = most significant half of word (digits 2^0 to 2^(-15)); **WS** = Working Store
Note: A *trigger* is an instruction or instruction-pair, the contents of a single word, to be acted on in order to set off a program process once the coding has been read into the computer.

5-bit Elliott telecode values used:

char	binary telecode	decimal value
*	00011	3
=	00110	6
(	10001	17
@	11000	24
'fs'	11011	27
'sp'	11100	28
'cr'	11101	29
'lf'	11110	30
'ls'	11111	31

digit	binary telecode	decimal value
0	10000	16
1	00001	1
2	00010	2
3	10011	19
4	00100	4
5	10101	21
6	10110	22
7	00111	7
8	01000	8
9	11001	25

alpha char	binary telecode	decimal value
E	00101	5
R	10010	18
O	01111	15

Each program line consists of a left column (in green) giving either the numeric address from which this location is reached, if not sequential, or the function or state operated on by the program at this point. This is followed by the store address (in red) and then the two instruction word. After that there is some explanation of what is being done here.

Ed: In the two pages above, Harry Lawrence found two anomalies in these basic standard library routines, whilst documenting the coding. It is remarkable how after over half a century, errors have come to light, which fortunately in these cases seem not to have mattered.

He reported:-

1. "I have completed the Fast Input Routine now but found that one of the values was incorrectly set! Well it still works, but is irrational as it doesn't deal precisely with the situation and must have been an error when originally written. It is the 7th line from the end which reads:

```
        0' 0 ,     0' 0/3     but should read:
        0' 0 , 110' 0/1.
```

It is trying to distinguish between a telecode "*" value 3 and a telecode "(" value 17.

The setting of 6 for comparison had me fooled for a while for that would be what is wanted to separate out both "*" and "=" from "(" which seems reasonable since "=" starts a sum-check line, but the order of lines is absolutely fixed and the whole thing would fail if a sum check line preceded the trigger line.

So the value to check against really is 3 and not 6 as noted above, but as I said, it still happens to work.

I checked to see if the constant 6 was used anywhere else, but it isn't, so it was just an error as the programmer obviously meant 3 and wrote 3, but it was punched in the right hand B-line position in error.

If you remember the TIs, and the form we used when coding, we would indicate when a number was simply to be punched on a line without punctuation by a vertical line on the left, just where the first function might have been. Well if it was left off or else missed by the punch-girl, then the result could be a 3 in the B-line position, and that translates to 6.2^-31".

2. "I think that the second instruction in line 180 is a mistake by the originator! I would have substituted "4'0" for the "0'0" written there, as that would enter the trigger with a clear accumulator instead of some meaningless indeterminate number that is based on the value of the number-generator at the time. However I thought that again I shouldn't fudge history by making a correction".