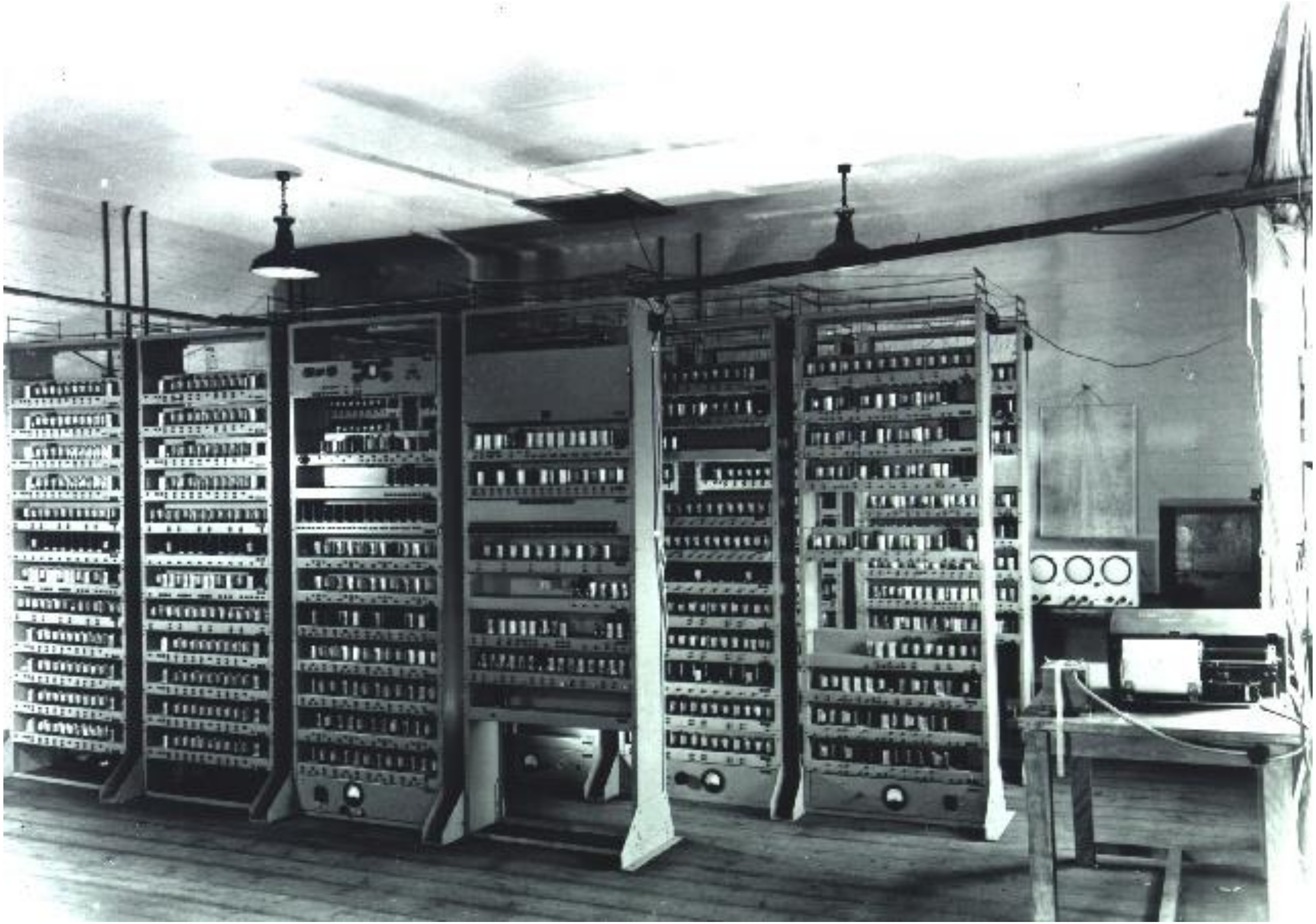
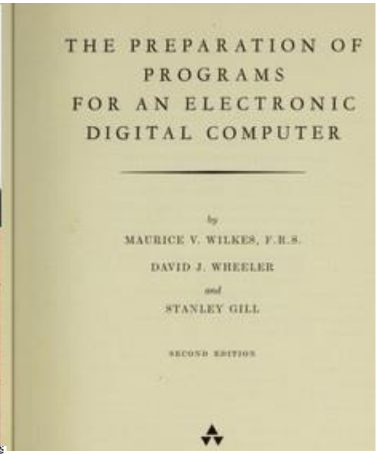
Programming the EDSAC

Andrew Herbert
The National Museum of Computing
16th November 2022

EDSAC FIRSTS

- The world’s first PRACTICAL electronic digital stored program computer = computer of the modern kind
- The world’s first computer programming system





THE PREPARATION OF PROGRAMS FOR AN ELECTRONIC DIGITAL COMPUTER

by

MAURICE V. WILKES, F.R.S.

DAVID J. WHEELER

and

STANLEY GILL

SECOND EDITION

The Preparation of Programs for an Electronic Digital Computer

Maurice .V. Wilkes, David .J. Wheeler and Stanley Gill

Addison Wesley, 1951

With special reference to the use of the EDSAC

Why a Programming System?

The methods of preparing programs for the EDSAC were developed with a view to reducing to a minimum the amount of labour required, and hence of making it feasible to use the machine for problems which require only a few hours of computing time as well as for those which require many hours. This necessitated the establishment of a library of subroutines and the development of systematic methods for constructing programs with their aid.

[WWG 1951]

Note emphasis on programmer productivity rather than on ”optimal programming”.

To the potential user of an automatic digital calculating machine, the successful design and construction of the machine itself is only a first step, though certainly an essential one. In order that the machine should in practice be useful to him in the calculations he may desire to carry out with its aid, the provision of an adequate organization for using the machine is as important as the machine itself.

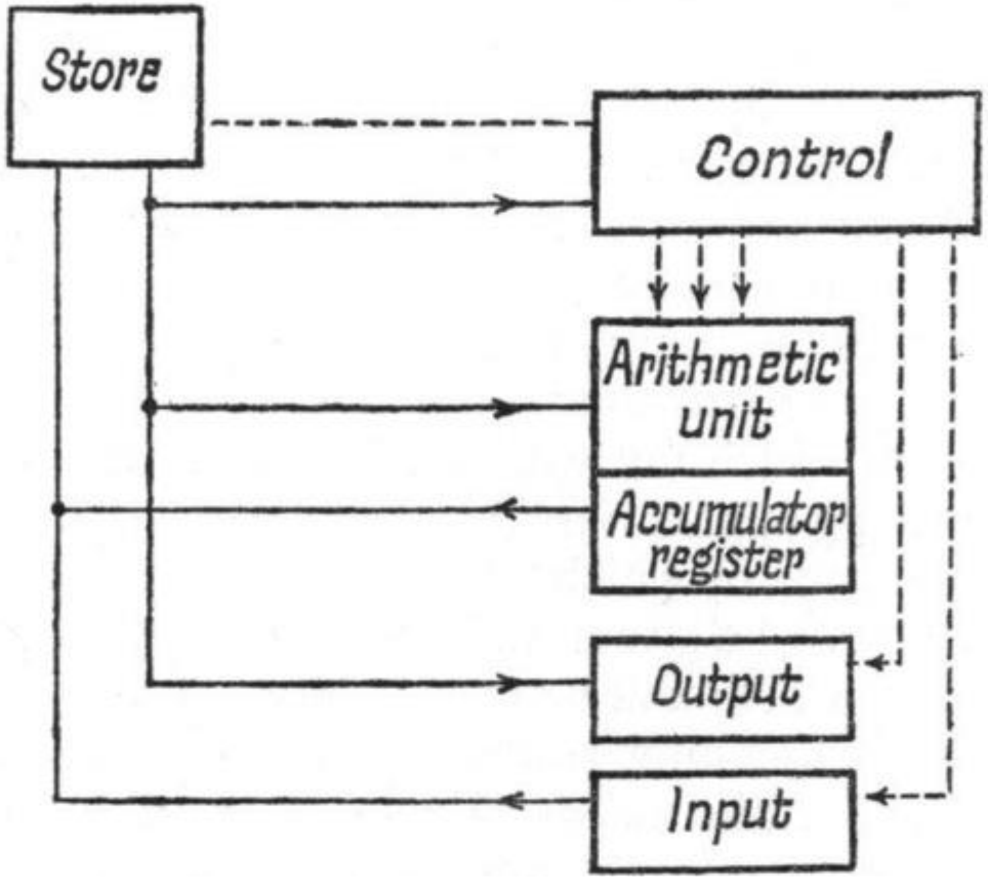
The process of building up [such] a library of subroutines, and testing its value by practical use, appears to have proceeded further at the Mathematical Laboratory of the University of Cambridge than elsewhere.

. . . it is a practical and useful system has been tested by experience; it divests programming of the appearance of being something of a magic art, closed except to a few specialists, and makes it an activity simple enough to be undertaken by the potential user who has not the opportunity to give his whole time to the subject.

The subject is one which is still developing. . .

EDSAC Block Diagram

1. Store: ultrasonic delay line holding 1024 × 17 binary digit numbers stored in true two's complement form, so most significant digit corresponds to the sign of the number.
2. Numbers are held in 1024 numbered "storage locations" numbered serially from 0 to 1023 for reference. Each such reference number is often called the "address" of the associated storage location.
3. Hence 17 bit numbers are often called "short numbers"
4. Two consecutive storage locations, starting from an even numbered address can be combined to make a 35 binary digit "long number".
5. Arithmetic unit: add, complement, collate, shift. Combine to enable subtract, multiply, round, but no divide.
6. Accumulator register of 71 bits.
7. Multiplier register of 35 bits.
8. Input: 5 hole paper tape read by photoelectric reader.
9. Output: teleprinter.
10. Control: an "order" passes from store into the control unit ("Stage I"), then it is executed {Stage II"). The machine then, generally, then automatically takes the next order from the location following that of the order just executed.



Order Code: O A F or O A D

O: Function code – operation to be carried out

A: Address of location to be used as operand

F or D: Short or Long number

A n	Add C(n) to Acc
S n	Subtract C(n) from Acc
H n	Copy C(n) to Multiplier
V n	Multiply C(n) by C(Mult) and add product to Acc
N n	Multiply C(n) by C(Mult) and subtract product from Acc
T n	Transfer C(Acc) to location n and clear Acc
U n	Transfer C(Acc) to location n but do not clear Acc
C n	Collate C(n) with C(Mult) and add to Acc
R D	Right shift Acc one place ($\times 2^{-1}$)
L D	left shift Acc one place ($\times 2$)

L 2^{p-2} F	Multiply by 2^p ($2 < p <= 12$)
R 2^{p-2} F	Multiply by 2^{-p} ($2 < p <= 12$)
E n F	If C(Acc) >= 0 execute next order from location n; otherwise proceed serially
G n F	If C(Acc) < 0 execute next order from location n; otherwise proceed serially
I n F	Read next 5 bit code from input to location n from tape reader
O n F	Print character set up on teleprinter, then set up m.s. 5 bits of location n as next character
F	Read back last set character
X	Ineffective – no-op
Y	Round Acc to 34 digits (i.e., add 2^{-35}).
Z	Halt and ring the bell

Fixed Point Arithmetic

Binary point assumed between top two most significant bits, so numbers are -1 <= x < 1.

Thus A order computes x+y-2 if x+y>=1, and x+y+2 if x+y<-1.

When two long numbers are multiplied together the resulting 69 digits are available in the Accumulator.

0.5 + 0.25 + (0.5 * 0.25) = 0.875

Compute x+y+xy; x in location 6, y in location 7.

(0)	T 8 F	
(1)	A 6 F	
(2)	A 7 F	
(3)	H 6 F	
(4)	V 7 F	
(5)	Z 0 F	
(6)	+0.5	01000000000000000
(7)	+0.25	00100000000000000
(8)	(spare)	

Demo1

Integer Arithmetic

Can treat accumulator as holding integers for A, S, C, L, R, E, G but for N, V multiplier is always treated as a fraction.

i.e., integers are stored as value * 2^{-16} so need to multiply by 2^{16} after multiplication.

10 + 5 + (10 * 5) = 65

Compute x+y+xy, x in location 8, y in location 9.

(0)	T 10 F	
(1)	H 8 F	
(2)	V 9 F	
(3)	L 512 F	; multiply by $2^{16} = 2^{11}*2^5$
(4)	L 8 F	
(5)	A 8 F	; note addition after
(6)	A 9 F	; multiplication
(7)	Z 0 F	
(8)	+10	00000000000001010
(9)	+5	00000000000000101
(10)	(spare)	

Demo2

Loops

- Loop to print digit 7 five times

- C(11) is "figure shift"
- C(12) is "7"
- C(13) is RET
- Loops while Acc < 0 (-5, -4, -3, -2, -1)

- Note: need to set figure vs. letter shift
- Note: output delayed one character

- Note: only G < 0 and E <= 0, but no "equals" order

(0)	T14F	
(1)	O10F	; Type figure shift
(2)	S9F	; Set count = -5
(3)	A10F	; Increment count
(4)	O12F	; Type '7'
(5)	G3F	; loop if count < 0
(6)	O13F	; Type RET
(7)	O11F	; Type figure shift
(8)	Z0F	; Halt
(9)	+5	
(10)	+1	
(11)	01011000000000000	
(12)	00111000000000000	
(13)	11000000000000000	
(14)	(spare)	

Demo3

Indexing

- Initially EDSAC had no index register.
- Invented for Manchester Mark 1 in 1949.
- Later adopted by EDSAC.

- So to do an indexed calculation, e.g., sum a vector, we have to write self-modifying code that manipulates program in store.

- To do arithmetic on orders we need to understand binary format of orders:
  - 5 m.s. bits: order code
  - 1 bit : spare (later add B register to address)
  - 10 bits: address in range 0-1023
  - l.s. bit: 0 = F, 1 = D

- Demo adds contents of vector starting at location 15

- We have to add +2 to location 2 each time around the loop to fetch the next element of table

(0)	T13F	; Clear acc
(1)	A14F	; Add sum
(2)	A15F	; Add vector[0] **
(3)	T14F	; Store in sum
(4)	A2F	; C(2)
(5)	A12F	; C(2)+2 (inc. address)
(6)	U2F	; Modify (2) **
(7)	S20F	; Check for end
(8)	G0F	; Loop if not at end
(9)	T13F	; Clear acc
(10)	A14F	; Result
(11)	Z0F	; Halt
(12)	+2	; address stride
(13)	+0	; workspace
(14)	+0	; sum
(15-19)	+1 +2 +3 +4 +5	; vector
(20)	A20F	; sentinel (Demo4)

Emulators etc

- Demos thus far run using EDSAC team test program generator:
  - Assembler written with "modern" facilities
  - Emulator written in C with tracing facilities
    - adapted from original by Lee Wittenberg
  - On EDSAC we use a "Signal Sequence Injector" box to set up program in main store from location 0 onwards
- Visit GitHub/andrewjherbert to find these:
  - edsacasm - https://github.com/andrewjherbert/edsacasm - Python
  - edsac - https://github.com/andrewjherbert/EDSAC-Emulator - C
- But this is not how EDSAC users wrote code...

UNIVERSITY MATHEMATICAL LABORATORY, CAMBRIDGE

EDSAC PROGRAMME SHEET

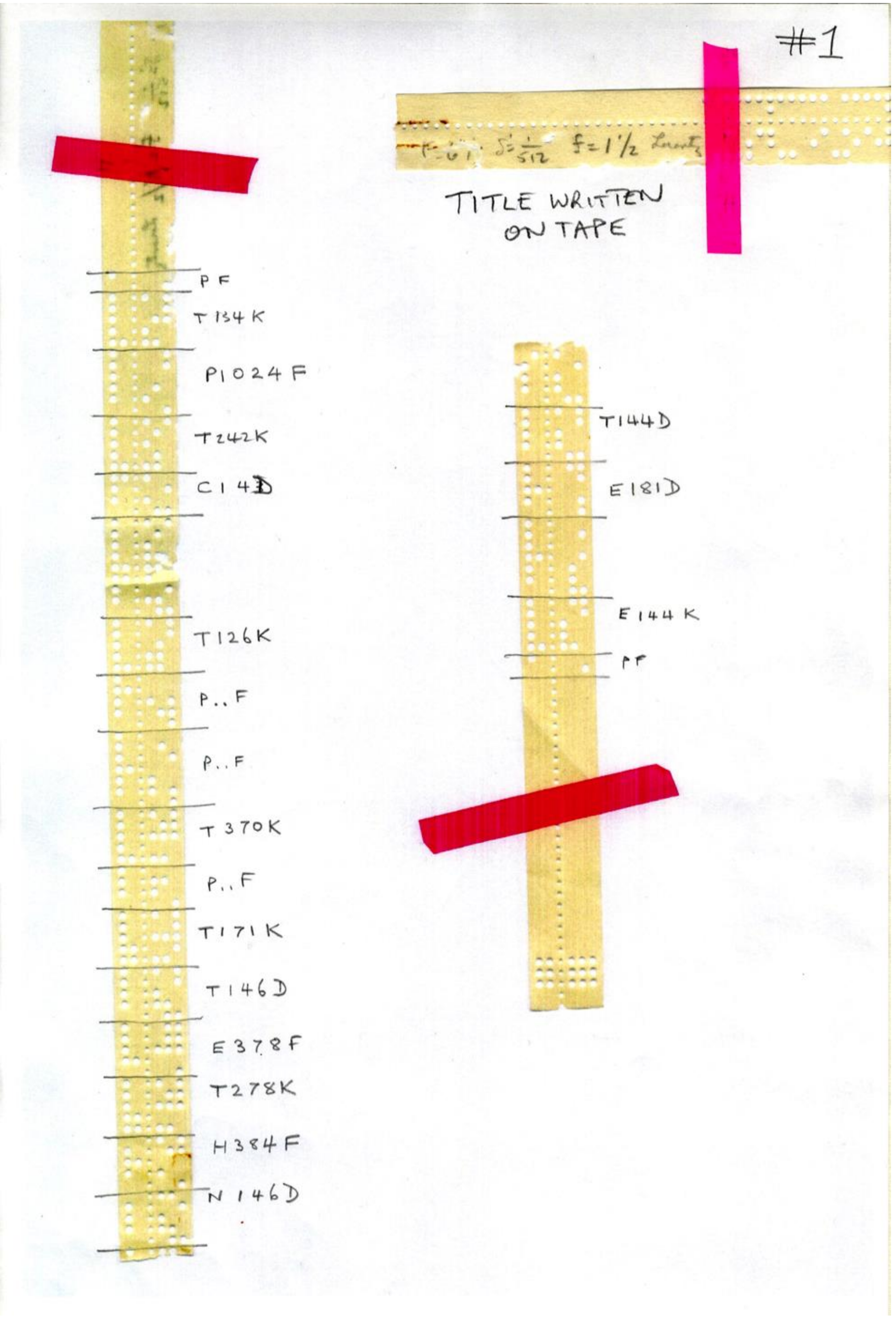
REF DATE

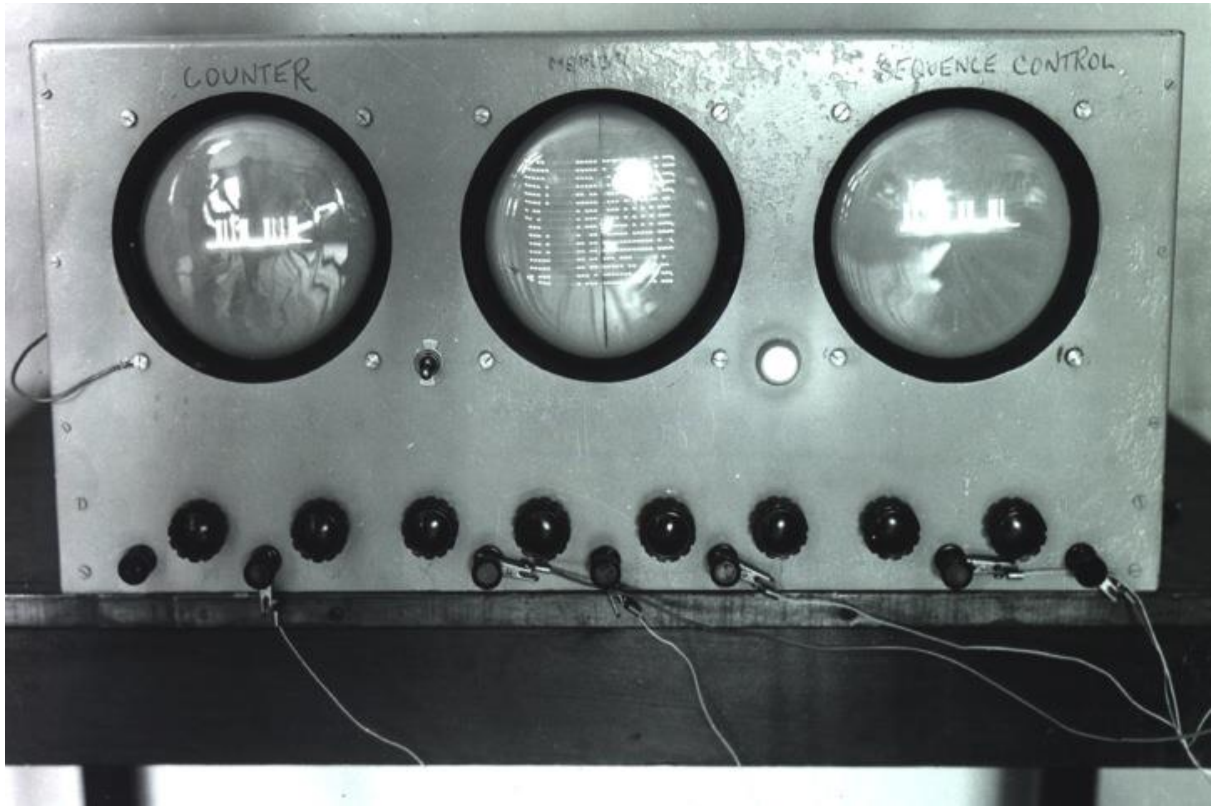
Calculation of curves for $y = \frac{1}{2}$ etc.

Calculates $\mu/2$, $\frac{\mu}{128 v_g}$, $\frac{1}{64 v_g}$

Use with tape WS6 :

	Order	Notes			Order	Notes
0	P F		this	0	V 2047 D	Starts at $\sin^2 D = -\varepsilon$ ($\varepsilon = 10^{-10}$)
1	T 134 K			1	K 4095 D	
2	P x F	$x = 4f \times 2048$	on this	2	P h (F; D)	$h = \sin^2 D \times 32768$ for starting value.
3	T 126 K	F in 126		3	P F	
4	P y F	$y = 2048\, F_0$		4	T 136 K	clears ν
5	T 294 K			5	P F	
6	E 231 F			6	P F	
7	T 231 K			7	T 358 K	
8	A 231 F			8	A 243 F	
9	G 165 F			9	T 126 K	
0	A D			0	T 323 K (20 F)	
1	T 288 D			1	T 171 K	
2	A 235 F			2	E 179 F	
3	G 91 F			3		
4	A 288 D			4	T 179 K	
5	T D			5	A 126 D	
6	O 241 F			6	T 132 K	
7	E 296 F			7	P 10813 F	
8	Ø F			8	P 32000 F	
9	W F	→ (Change to P b F where $b = \Delta \sin^2 D \times 32768$ for $\sin^2 D$ in other steps)		9	T 317 K	
0	P ~~512~~ g F	$\Delta F \times 2048 = g\left(x = \frac{E}{f^2}\right)$		0	P 256 F	O 241 F
1	T 329 K			1	T 314 K	T D
2	A 126 D			2	T 36 D	E 316 F
3	A 243 F			3	E 248 F	T 366 K
4	T 126 D	n′ = no of steps before next value of $\sin^2 D$		4	T 248 K	P n′ F
5	T 355 K			5	H 36 D	T 211 K } for X only.
6	A 128 D			6	V 288 D	S D
7	A 242 F			7	T D	
8	T 128 D			8	A 251 F	E 144 K
9	T 128 K.			9	G 91 F	P F.




COUNTER
SEQUENCE CONTROL

$\beta = \frac{7}{16}$

$\delta y = \frac{3}{56}$

β

$\beta = \frac{7}{16}$

$\delta y =$

	031164²	+019084
	+032194	+018876
	+033509	+018732
	+035237	+018687
	+037588	+018794
	+040919	+019138
1	+045875	+019864
	+053687	+021222
	+066909	+023656
	+091337	+027966
	+141844	+035461
	+268904	+047536

	+031130	+019063
	+032182	+018868
	+033528	+018743
	+035304	+018723
	+037726	+018863
	+041163	+019252
1.05357..	+046278	+020039
	+054331	+021476
	+067907	+024009
	+092799	+028414
	+143687	+035922
	+270334	+047789

	+031106	+019048
	+032179	+018867
	+033558	+018759
	+035380	+018763
.10714...	+037871	+018936
	+041412	+019369
	+046680	+020213
	+054959	+021724
	+068860	+024346
	+094165	+028832
	+145354	+036339
	+271576	+048008

	+031091	[illegible]039
	+032185	+018870

Initial Orders

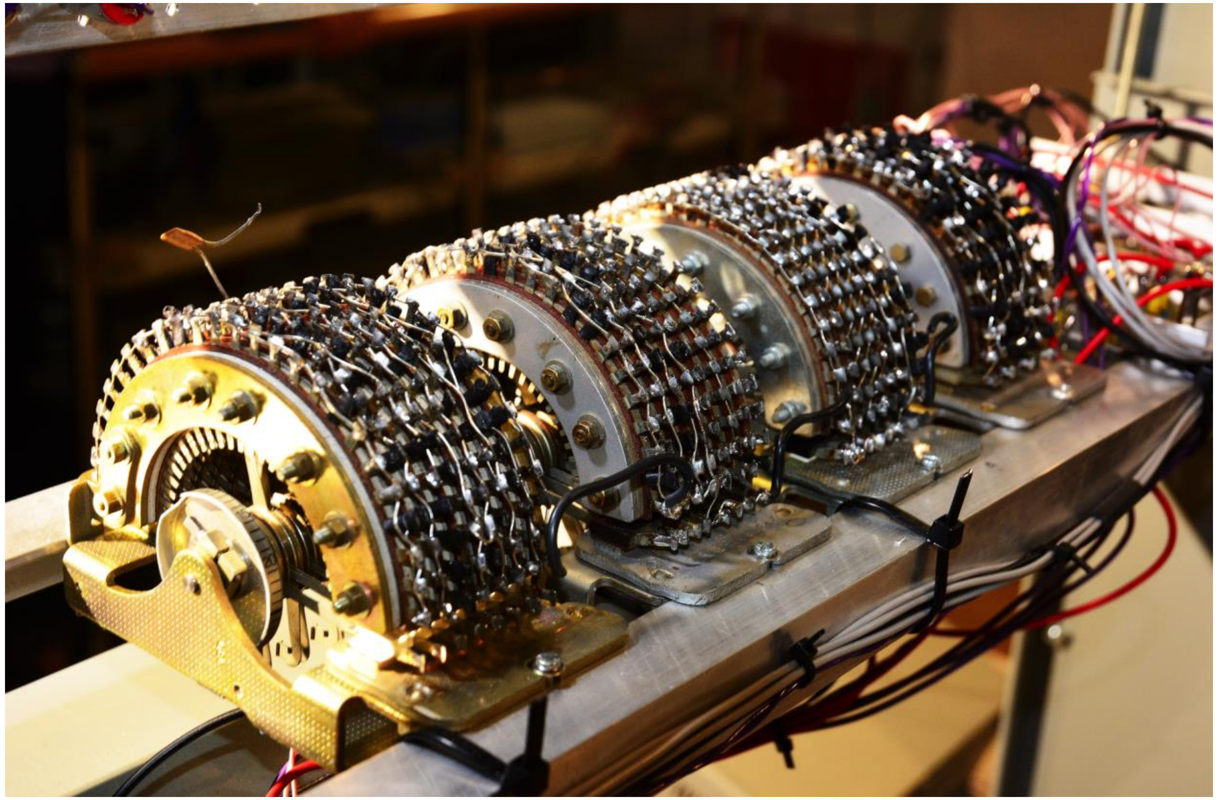
Fixed program to load source programs from paper tape into store

Input is alphanumeric

Combined assembler and linker to enable user code to be linked to predefined library routines

Unique to EDSAC

Programming tour de force by David Wheeler



Proc. Royal Society A, 202, August 1950: D.J. Wheeler, Programme organization and initial orders for the EDSAC. https://royalsocietypublishing.org/doi/10.1098/rspa.1950.0121

Initial orders concepts

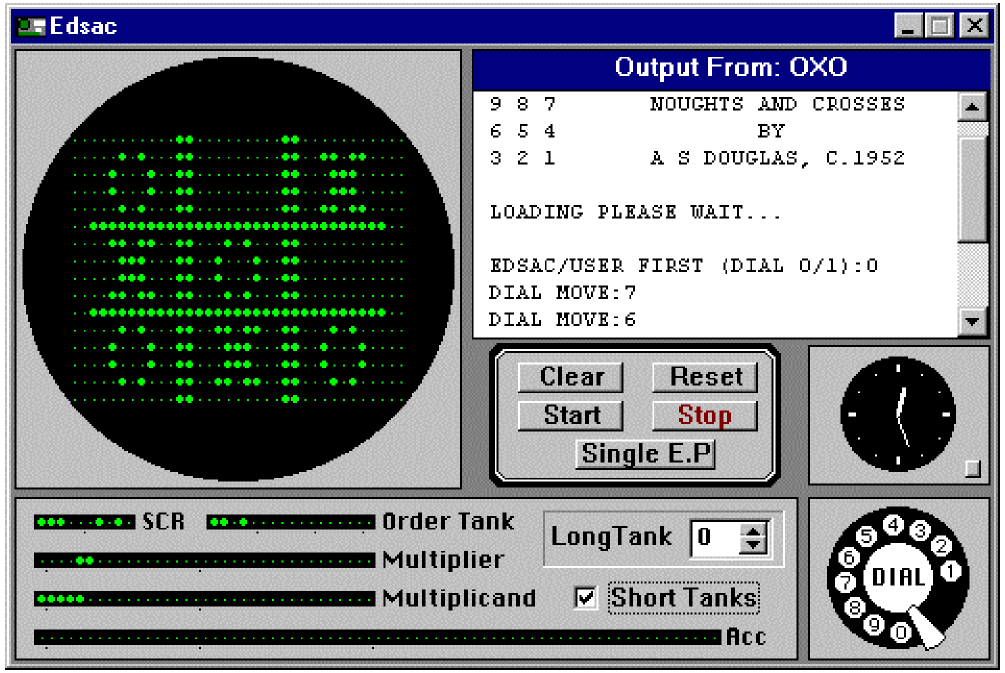
- Instructions in alphanumeric form rather than binary
  - Like modern assembly code
- Control codes to direct initial orders where to load and how to fix up addresses, start execution
  - To enable linking in subroutines in arbitrary order
- Addressing relative to a previously set parameter (control code)
- But no error handling!

Warwick Simulator

Written by Martin Campbell-Kelly

Available for Windows and MacOS

GUI replicates original EDSAC operation



https://edsac.net

Includes Tutorial Guide, original EDSAC subroutine library, worked example programs

Hello world

T 64 K G K and E Z P F are
 control combinations

θ is a “parametric address”

T 64 K – load from loc 64

G K – set **θ** (to 64)

E Z P F – enter program at

Location **θ** (64)

		T	64 K	Load from location 64
		G	K	Set θ parameter
Start →	0	Z	F	Stop
	1	O	5 θ	Print letter shift
	2	O	6 θ	Print "H"
	3	O	7 θ	Print "I"
	4	Z	F	Stop
	5	*	F	Letter shift
	6	H	F	"H"
	7	I	F	"I"
		E	Z	Enter at location 0θ
		P	F	

(a) Program text

```
T64K
GK
ZF
O5@
O6@
O7@
ZF
*F
HF
IF
EZPF
```

(b) Program tape

* Is erase character (32 decimal)

N.B. Data input as instructions

Demonstration Programs/Hello.txt

EDSAC character codes

Programs prepared on perforator

Note NO figure / number shift

Output produced in Teleprinter code

Order field in instruction is the bit pattern of the order character, i.e., A = 11101

Note convention for typing Greek letters when using emulators

Table 2 Edsac Character Codes

Perforator		Teleprinter		Binary	Decimal
Letter shift	Figure shift	Letter shift	Figure shift		
P	0	P	0	00000	0
Q	1	Q	1	00001	1
W	2	W	2	00010	2
E	3	E	3	00011	3
R	4	R	4	00100	4
T	5	T	5	00101	5
Y	6	Y	6	00110	6
U	7	U	7	00111	7
I	8	I	8	01000	8
O	9	O	9	01001	9
J		J		01010	10
p		Figure Shift		01011	11
S		S	"	01100	12
Z		Z	+	01101	13
K		K	(	01110	14
Erase[1]		Letter Shift		01111	15
Blank tape[2]		(no effect)		10000	16
F		F	$	10001	17
q		Carriage Return		10010	18
D		D	;	10011	19
f		Space		10100	20
H	+	H	£	10101	21
N	-	N	,	10110	22
M		M	.	10111	23
D		Line Feed		11000	24
L		L	)	11001	25
X		X	/	11010	26
G		G	#	11011	27
A		A	-	11100	28
B		B	?	11101	29
C		C	:	11110	30
V		V	=	11111	31

Notes

1 Erase is represented by an asterisk ("*") in the simulator. When this character is *output*, it sets the teleprinter into letter shift.
2 Blank tape is represented by a period ("."). This character has no effect on output.
3 The personal computer text environment has only a "newline" character. On the Edsac simulator, the line-feed character is interpreted as a newline character, and carriage returns are thrown away.
4 The symbols q, f , D or p are typed as @, !, & and #, respectively.

Control combinations

T m K	set load point to m
G K	set $\boldsymbol{\theta}$ parameter to load point
T Z	restore $\boldsymbol{\theta}$ parameter
E m K P F	enter program at location m
E Z P F	enter program at location $\boldsymbol{\theta}$
P Z or P K	start of new tape block

Subroutines – the Wheeler jump

A m F to pick up where calling from (m+1)

A m F is 11000... so negative

C(3) = U 2 F

Calculate E m+2 F and store as final instruction

Return to caller

```
m    | A  m  F | pick up self              ┐
m+1  | G  n  F | jump to subroutine        │ master routine
m+2  |    .    | control returns here      ┘
     |    .    |
     |    .    |
n    | A  3  F | form return link          ┐
n+1  | T  p  F | plant return link         │
     |    .    |                           │ subroutine
     |    .    |                           │
p    | (  .  ) | return link planted here  ┘
```

Cubes

Nichomacus' formula for cubes:

$1^3 = 1$

$2^3 = 3 + 5$

$3^3 = 7 + 9 + 11$

$4^3 = 13 + 15 + 17 + 19$

etc

Use library routine P6 to print integers

.. represents blank tape

	G	K	Set θ-parameter
Enter→ 0	Z	F	Stop
1	O	29 θ	Figure shift
22 → 2	O	30 θ	New line
3	O	31 θ	
4	A	23 θ	k to 0F
5	T	F	
6	A	6 θ	Print 0F using P6
7	G	56 F	
P6 → 8	T	23 θ	Zero to k
9	A	24 θ	n+1 to n
10	A	27 θ	
11	T	24 θ	
12	S	24 θ	-n to count
21 → 13	T	26 θ	
14	A	25 θ	m+2 to m
15	A	28 θ	
16	U	25 θ	
17	A	23 θ	k+m to k
18	T	23 θ	
19	A	26 θ	Increment count
20	A	27 θ	
21	G	13 θ	Jump to 13 if count • 0
22	E	2 θ	Repeat main cycle
23	P	D	k (n^3; =1 initially)
24	P	D	n (=1 initially)
25	P	D	m (=1 initially)
26	P	F	count
27	P	D	=1
28	P	1 F	=2
29	π	F	figs
30	θ	F	cr
31	Δ	F	lf

(a) Master routine

Routine	Location of first order	Number of storage locations occupied
P6 (print)	56	32
Master	88	-

(b) Table of routines

space P K

T 56 K

$\boxed{\text{P6}}$

space P Z

$\boxed{\text{Master}}$

E Z P F

(c) Make-up of program tape

1
8
27
64
125
216
343
512
.
.
.

(e) Printout

```
[Cubes]
..PK
T56K
[P6]
GKA3FT25@H29@VFT4DA3@TFH30@S6@U1F
V4DU4DAFG26@TFTFO5FA4DF4FS4F
L4FT4DA1FS3@G9@EFSFO31@E20@J995FJF!F
..PZ
[Cubes Master]
GK
ZF
O29@
O30@
O31@
A23@
TF
A6@
G56F
T23@
A24@
A27@
T24@
S24@
T26@
A25@
A28@
U25@
A23@
T23@
A26@
A27@
G13@
E2@
PD
PD
PD
PF
PD
P1F
#F
@F
&F
EZPF
```

(d) Program tape

Demonstration Programs/Cubes.txt

Notes

Conventional “coding sheet” style for writing programs

No layout on EDSAC tape

No comments on EDSAC tape

Use of $\boldsymbol{\theta}$ to make code position independent

Constants written as pseudo orders

Advanced features

Code letters:

Code-letter	*Location*	*Value*
F	41	0
q	42	Origin of current routine
D	43	1
f , H, N, M … V	44, 45, 46 … 55	For use by programmer

Used to create position independent code and data cross references

Subroutine parameters:

Pass via fixed address (often 0)

Include in calling sequence

Run and delete open subroutines on the fly to save store...

	G	K	
	T	47 K	Set M parameter
	P	21 θ	
	T	Z	
0	S	1 M	Set count to -9
19 → 1	T	6 M	
2	A	2 M	$1 \cdot 2^{-4}$ to 0D
3	T	D	
4	A	7 M	$n \cdot 2^{-4}$ to 0F
5	T	4 D	
6	A	6 θ	Set 0D to 0D/4D(ie. 1/n)
7	G	56 F	using subroutine D6
D6 → 8	O	3 M	Output new line
9	O	4 M	
10	O	5 M	Output decimal point
11	A	11 θ	Print 0D
12	G	92 F	using subroutine P1
13	P	10 F	Parameter for P1
P1 → 14	A	7 M	Increment n
15	A	2 M	
16	T	7 M	
17	A	6 M	Increment
18	A	M	and test counter
19	G	1 θ	
20	Z	F	Stop
M 0	P	D	= 1
1	P	4 D	= 9
2	Q	F	$= 1 \cdot 2^{-4}$
3	θ	F	carriage return
4	Δ	F	line feed
5	M	F	decimal point
6	P	F	count
7	W	F	$= n\ (=2 \cdot 2^{-4}$ initially)

(a) Master routine

space P K

T 56 K

M3

θΔ*RECIPROCALSθΔπ Table heading

space P Z

T 56 K

D6

space P Z

P1

space P Z

Master

E Z P F

(c) Make-up of program tape

Routine	Location of first order	Number of storage locations occupied
D6 (divide)	56	36
P1 (print)	92	21
Master	113	-

(b) Table of routines

RECIPROCALS

.4999999999
.3333333333
.2499999999
.2000000000
.1666666666
.1428571428
.1249999999
.1111111111
.0999999999

(d) Printout

Load at loc 56
Print caption
Load at loc 56
Divide subroutine
Print subroutine
Main program
Enter main program
M block at 21 - data

Reciprocals Folder/Reciprocals.txt

• Debugging – post-mortem

Start reciprocals

. . . Executes . . .

Start PM5

Dial start location, e.g., 134 (113+21 = start of data)

Debugging – Checking (i.e., tracing)

Assemble program with checking routine at end

C7 – execution trace

```
.
.
.
space P Z

Master

space P Z
G K T 45 K P F
P 113 F
PNDqPN

C7
```

C10 – arithmetical trace

```
.
.
.
space P Z

Master

space P Z
GKT45KP37qP10F
P 113 F
PNDqPN

C10

E 113 K P F
```

Using command line emulator

Demo5

punch – convert ASCII to EDSAC code

    same conventions as Warwick emulator for special symbols etc

edsac – run emulator taking input from stdin

    -v1/-v2 tracing

    -lnnn order limit

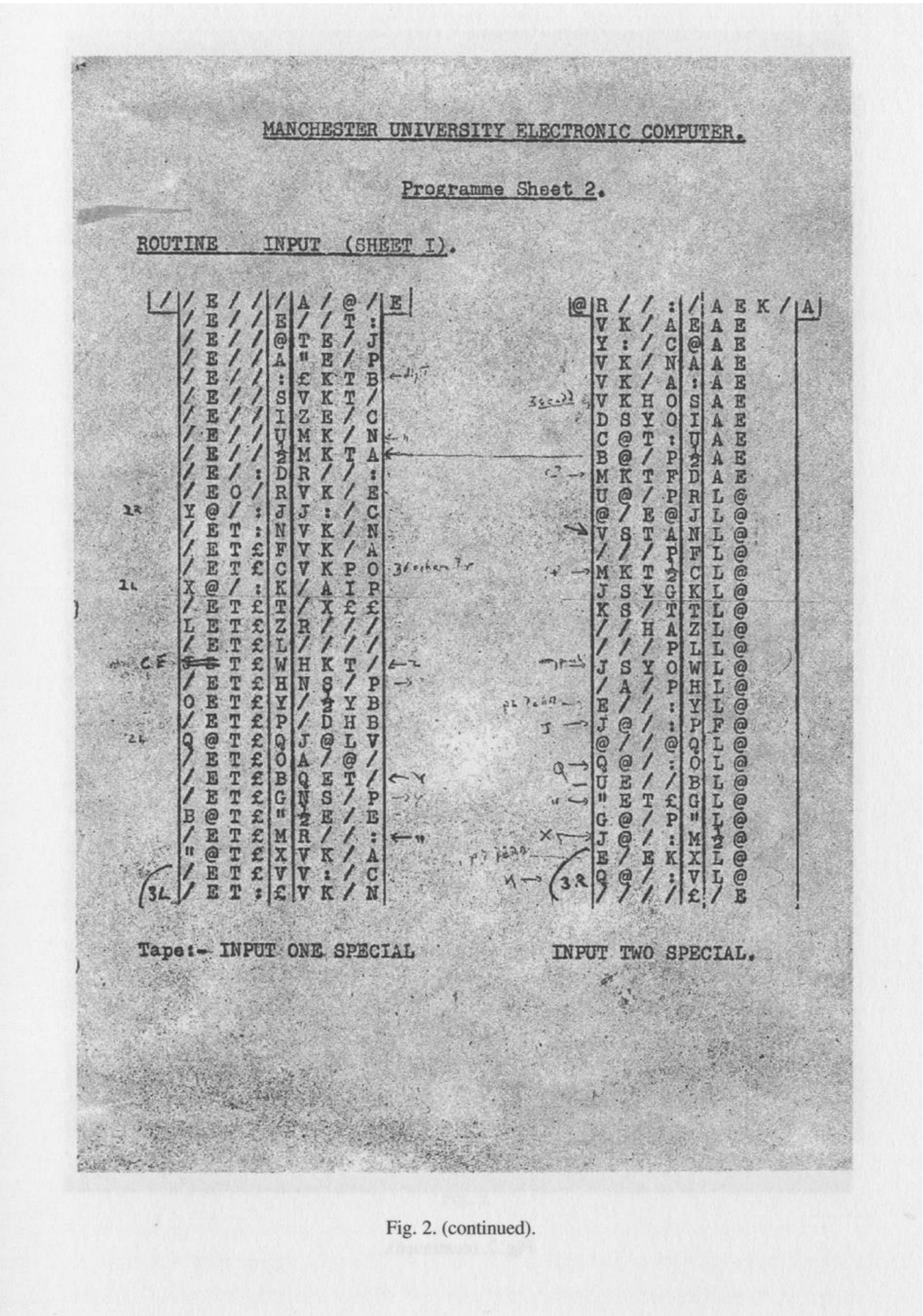
    -s to start

    -b for EDSAC replica SSI emulation

tprint – convert Teleprinter output to UTF

By contrast…

From Turing's programming guide for Manchester Mark 1



MANCHESTER UNIVERSITY ELECTRONIC COMPUTER.

Programme Sheet 2.

ROUTINE INPUT (SHEET I).

```
/ | / E / / | / | A / @ / | E
  | / E / / | E | / / T :
  | / E / / | @ | T E / J
  | / E / / | A | " E / P
  | / E / / | : | £ K T B
  | / E / / | S | V K T /
  | / E / / | I | Z E / C
  | / E / / | U | M K / N
  | / E / / | ½ | M K T A
  | / E / : | D | R / / :
  | / E O / | R | V K / E
  | Y @ / : | J | J : / C
  | / E T : | N | V K / N
  | / E T £ | F | V K / A
  | / E T £ | C | V K P O
  | X @ / : | K | / A I P
  | / E T £ | T | / X £ £
  | L E T £ | Z | R / / /
  | / E T £ | L | / / / /
  | / E T £ | W | H K T /
  | / E T £ | H | N S / P
  | O E T £ | Y | / ½ Y B
  | / E T £ | P | / D H B
  | Q @ T £ | Q | J @ L V
  | / E T £ | O | A / @ /
  | / E T £ | B | Q E T /
  | / E T £ | G | N S / P
  | B @ T £ | " | ½ E / E
  | / E T £ | M | R / / :
  | " @ T £ | X | V K / A
  | / E T £ | V | V : / C
  | / E T : | £ | V K / N
```

```
@ | R / / : | / | A E K / | A
  | V K / A | E | A E
  | Y : / C | @ | A E
  | V K / N | A | A E
  | V K / A | : | A E
  | V K H O | S | A E
  | D S Y O | I | A E
  | C @ T : | U | A E
  | B @ / P | ½ | A E
  | M K T F | D | A E
  | U @ / P | R | L @
  | @ / E @ | J | L @
  | V S T A | N | L @
  | / / / P | F | L @
  | M K T ½ | C | L @
  | J S Y G | K | L @
  | K S / T | T | L @
  | / / H A | Z | L @
  | / / / P | L | L @
  | J S Y O | W | L @
  | / A / P | H | L @
  | E / / : | Y | L @
  | J @ / : | P | F @
  | @ / / @ | Q | L @
  | Q @ / : | O | L @
  | U E / / | B | L @
  | " E T £ | G | L @
  | G @ / P | " | L @
  | J @ / : | M | ½ @
  | E / E K | X | L @
  | Q @ / : | V | L @
  | / / / / | £ | / E
```

Tape:- INPUT ONE SPECIAL   INPUT TWO SPECIAL.

Fig. 2. (continued).

How to get started

- Download Warwick simulator, work through examples
- Pitfalls:
  - Remember the store is tiny
  - Be careful about long versus short numbers.
  - Remember to scale calculations.
  - Remember no index registers so vectors, arrays and stacks tedious to manipulate – consider writing subroutines / interpreters
  - Read library subroutine specifications carefully to understand parameter passing conventions and any special control combinations to load them.
  - Use code letters to divide code and data into short blocks to avoid having to renumber addresses if additional code or data inserted (or deleted).
  - Beware miscoding pseudo orders (i.e., constants)
  - Must use library routines (R series) to input long numbers – n.b., R2 will input long integers at load time
  - Remember need to set teleprinter shift and to force out last character