Instruction sets and instruction times for the Elliott 152, 153 and Nicholas computers.

Note: all references are listed in section E1/X5.

Instruction set(s) and instruction times for the Elliott 152.

It has not so far proved possible to discover a definitive account of the 152's instruction set. (The surviving Borehamwood reports are dominated by accounts of the design and control of the advanced MRS5 radar system). The 152's instruction set would have been somewhat similar in principle to that of the Elliott 153 (see later), since the register-level structure of the two machines is similar. The 152's instruction length was, however, 20 bits whereas the 153's length was 64 bits.

When programming the Elliott 152, each ROM slide held 16 instructions. The instruction format would seem complex to modern eyes, since it gave the programmer the sort of low-level control normally associated with microprogramming. The gunnery control programs, once written, were deliberately unalterable, so that the considerable investment of initial programmer-effort was a one-off event.

Numbers were represented in the 152 as two's complement fractions, but with the implied point being two positions from the most-significant end. This system, tailored to the representation of sine and cosine functions, provided a guard-digit as an overflow indicator.

The control cycle of the Elliott 152 was based upon a 'beat' of 60 microseconds, corresponding to 16 of the 3-microsecond digit-periods plus a four-digit gap. All arithmetic operations could be performed within this 60-microsecond period. Remember, however, that more than one arithmetic task could be performed concurrently within a single instruction-time (see section E1X2). Therefore, the 152's effective speed was faster than might be supposed from an apparent rate of one instruction per 60 microseconds.

Instruction set(s) and instruction times for the Elliott 153 (the DF computer).

In conventional terms, the 153 has two functional units: an Accumulator (a short-hand way of describing an arithmetic/logic unit, ALU) and a fast Multiplier. Both units can operate concurrently on 16-bit fixed-point fractional operands, using what would now be called a single-address instruction format. However, the actual programming of the 153 necessitates the setting of individual bits in a 64-bit instruction that is more reminiscent of a microprogram word.

The four 16-bit words making up each instruction for the Elliott 153 are partitioned as shown diagrammatically below. The following abbreviations are used:

s	spare, ie unallocated, bit(s)
S1	RAM store number 1
S2	RAM store number 2
S3	a 16-bit constant, contained in the fourth word of an instruction
Input/Output:	The Elliott 153 has ten paper tape readers for input and three teleprinters for output. These devices can be individually selected and activated by digits 10 – 15 of instruction-word 1.
J	a *Jump if* indicator, used in conjunction with the three Control bits in the third word of an instruction.
Md	multiplicand register
Mr	multiplier register
Acc	the main accumulator
Acc F	the function (op code) for the main ALU. These two bits specify one of the following basic operations: Add, Subtract, Reverse-subtract, logical AND.
Task	These two bits control the interrogation of certain *Task selector* and *Station rejector* switches in the operator's console – see later.
Control	These three bits have two distinct uses. When J = 1, they define the possible conditions for transfer of control. The possibilities are: *jump if acc negative* or *jump if acc zero* or *jump unconditionally*. If J = 0, the three Control bits are used for disc transfers, the possibilities being the three commands to *Set Relay Tree, Read from disc,* and *Write to disc.* The detailed action is described later.

2	4	3	6	1
s	**S1 addr**	**S1 in**	**Input/output select**	**J**

2	4	3	3	3	1
s	**S2 addr**	**S2 in**	**Md in**	**Mr in**	*s*

3	3	2	3	2	3
s	**Acc in 1**	**Acc F**	**Acc in 2**	**Task**	**Control**

16
S3, a constant (ie literal)

The format for the four 16-bit words that make up a 64-bit instruction for the Elliott 153 computer.

The detailed allocation of particular groups of bits in the four words shown above is now given, based on information contained in reference 1. For convenience, the bits are numbered 1 – 16, starting for convenience at the left-hand (most-significant) end.

Word 1.

Bits 7 – 9: source of the data for RAM cache S1.

000	regenerate (see note (a).
100	input from the multiplier register, M

010	input from S2
110	input from the selected Tape Reader
001	input from S3, the constant specified in the present instruction.
101	input from the accumulator, A
011	input = 2A (ie acc shifted one place left)
111	input = A/2 (ie acc shifted down one right)

Bits 10 – 12: Tape Readers and Teleprinters.

000	no action
100	read from Tape Reader 6
010	read from Tape Reader 7
110	read from Tape Reader 8
001	read from Tape Reader 9
101	read from Tape Reader 10, and step on all Readers.
011	Teleprinter 1
111	Teleprinter 3

Bits 13 – 15: Tape Readers and Teleprinters (continued)

000	no action
100	Interrogate Tape Readers
010	Teleprinter 2
110	read from Tape Reader 1
001	read from Tape Reader 2
101	read from Tape Reader 3
011	read from Tape Reader 4
111	read from Tape Reader 5

Word 2.

Bits 7 – 9: source (routing) for RAM cache S2.

000	regenerate (see note (a).
100	input from multiplier register, M
010	input from Tape Readers
110	input from S1
001	input from S3
101	input from the accumulator, A
011	input = 2A (ie acc shifted one place left)
111	input = A/2 (ie acc shifted down one place right)

Bits 10 – 12: source (routing) for the multiplicand.

000	input is unity
100	input from multiplier register, M
010	input from S2
110	input from S1
001	input from S3
101	input from the accumulator, A
011	input = 2A (ie acc shifted one place left)
111	input = A/2 (ie acc shifted down one place right)

Bits 13 – 15: source (routing) for input to the multiplier.

000	retain the previous number in the M register

100	input from multiplier register, M
010	input from S2
110	input from S1
001	input from S3, the constant specified in the present instruction.
101	input from the accumulator, A
011	input = 2A (ie acc shifted one place left)
111	input from Programme Selector (on the operator's console (control desk))

Word 3.

Bits 4 – 6: source (routing) of ALU (accumulator) input 1.

000	zero
100	(spare, not used)
010	input from S2
110	input from S1
001	input from S3, the constant specified in the present instruction.
101	input from multiplier register, M
011	input = 2A (ie acc shifted one place left)
111	(spare, not used)

Bits 7 – 8: accumulator function (op. code).

00	ADD:	input 1 + input 2.
10	SUB:	input 1 – input 2.
01	REVSUB:	input 2 – input 1
11	AND:	input 1 & input 2 (known originally as *Collate*).

Bits 9 – 11: source (routing) of ALU (accumulator) input 2.

000	input from the accumulator, A
100	input from the number set up on the control-desk handkeys
010	input from S2
110	input from S1
001	input from S3, the constant specified in this instruction.
101	input from the multiplier register, M
011	input = 2A (ie acc shifted one place up)
111	input = 1/2A (ie acc shifted down one place)

Bits 12 – 13: operator's control console - numbers routing unit.

00	Task selector, switches 1.
10	Station rejector, switches 2, and Task Reset.
01	Station rejector, switches 1
11	Task selector, switches 2, and Tape Self Drive (?)

Bits 14 – 16: control for this instruction.

000	nil (no action)
100	CT negative: jump to address in M if accumulator register is negative
010	Read from disc, using track/location address held in accumulator
110	Mix S2 (see below)
001	Write to disc
101	Set relay tree, in preparation of disc read/write
011	CT zero: jump to address in M if accumulator register is zero.
111	Mix S1 (see below)

Blocks of data may be read from disc into S2, or written from S1 to disc, by means of reading, writing and disc-addressing commands. The desired disc-address is first calculated in the accumulator and then set up on a relay-tree. The three commands to Set Relay Tree, Read from disc, and Write to disc, are activated by three combinations of bits 14 to 16 in instruction-word 3. Three further combinations of bits 14 to 16 are used for control transfers (see below). The remaining two combinations of bits 14 to 16 are used in the calculation of dynamic addresses, as follows. An address calculated by a previous instruction and placed in the accumulator can be used during the present instruction to address either RAM S1 or RAM S2 – (thus replacing the store-addresses otherwise specified by bits 3- 6 of instruction-words 1 or 2).

An example of a complex Elliott 153 instruction.
A typical Elliott 153 instruction has more potential for useful work than does a typical instruction seen on other 1950s machines – or indeed on most modern computers. The following illustrative example is given in reference 1. A single 153 instruction can be used to achieve the following sequence of operations in one machine cycle (72 microseconds):

a) Read a character from tape reader 6 and store this in address 12 of S1;
b) Transfer the previous contents of the above address to location 5 of S2;
c) Multiply the previous contents of location 5 in S2 by π/4, a constant contained at S3 within the present instruction, holding the result in the multiplier register;
d) Add the previous contents of the multiplier register to the previous contents of location 5 in S2 and hold the result in the accumulator.

In a modern register-set computer having four working registers, R1 – R4, and an accumulator, the above sequence would have to be programmed in about five separate instructions as follows:

```
R3  := π/4;
R4  := R2 * R3;
ACC := R2 + R4;
R2  := R1;
R1  := char read from PTR6.
```

In conclusion, it is difficult to compare the 153's architectural power with conventional computer designs implemented in similar 1950s technologies. At one extreme, one 72-microsecond instruction on the 153 may do the work of about a dozen simpler instructions each taking 24 microseconds. At the other extreme, one 72-microsecond instruction is about three times slower in performing very simple operations.

Instruction set & instruction times for the Elliott Nicholas.
Two 16-bit instructions are packed into each 32-bit word. Since addressing is to word boundaries, it is not possible to jump to an 'odd' instruction. Each instruction assigns 6 bits for the function (op.code) and 10 bits for the operand address.

The format for a pair of instructions is as follows, where the right-hand (less-significant) one is executed first:

<-- ***second instr***. --><--- ***first instr***. - -->

6	10	6	10
F	**N**	**F**	**N**
Op code	Address	Op code	Address

The Function bits (op code) are divided, for hardware convenience, into two fields as shown in the lists below. Each field may be treated as an octal digit to which the Nicholas Assembler assigns an alphabetic letter. Although the two fields might be imagined as providing a maximum of 64 instructions, far fewer than this have any practical use. The most useful surviving description of the valid instructions will be found in reference 5 but even this document is silent about many combinations of the F bits.

The following (modern) notation is used in describing Nicholas instructions.

a = contents of the accumulator
x = contents of storage location n
t = the current five-bit character as read in from paper tape, placed as the least-significant five bits in a 32-bit word.

Primes are used to indicate the new values at the conclusion of an instruction. Operand addresses are denoted by n; dummy addresses in monadic instructions are denoted by m.

The Nicholas instruction set is composed of combinations of the following two groups of basic operations.

F bits	***action***		***corresponding Assembler letter***
- - - 000	do nothing		O
- - - 001	a’ = a + x	(add)	A
- - - 010	jump to n if a < 0		Z
- - - 011	unconditional absolute jump		T
- - - 100	special action (see below)		Y
- - - 101	a’ = a + t	(input a character)	I
- - - 110	a’ = a x x	(multiplication)	M
- - - 111	x’ = a	(write acc to memory)	W
000 - - -	do nothing		O
001 - - -	a’ = 0	(clear acc)	C
010 - - -	a’ = 2 x a	(arith shift left one place)	D
011 - - -	a’ = a/2	(arith shift right one place)	H
100 - - -	special action (see below)		X
101 - - -	a’ = – a	(negate acc)	N
110 - - -	a’ = a & x	(logical AND)	L
111 - - -	a’ = a – x	(subtract)	S

The actual bit-pattern for the instruction *Add the contents of location 13 to the accumulator* would be: 0000010000001101. In Nicholas Assembler, this would be written as: A 13.

The Nicholas Assembler, known as the *Translation Input,* introduced an additional single-letter mnemonic, P, which was a pseudo-instruction translated by software into a call to the library subroutine for printing, as described below.

Further valid Nicholas instructions may be composed of certain combinations of the above two lists of basic hardware actions. Here are the more useful combinations:

F bits	***action***		***corresponding Assembler letters***
001001	a' = x	clear and add (ie load)	CA
010001	a' = 2a + x	double and add	DA
011001	a' = a/2 + x	halve and add	HA
101001	a' = x – a	reverse subtract	NA
001111	x' = a; a' = 0	store acc, then clear acc	WC
010111	x' = a; a' = 2a	store acc, then double acc	WD
011111	x' = a; a' = a/2	store acc, then halve acc	WH
101111	x' = a; a' = - a	store acc, then negate acc	WN
111111	x' = a; a' = a – x	store acc, then subtract old x	WS
110111	x' = a; a' = a & x	store acc, then AND with old x	WL
001101	a' = t	read an input character; step the paper tape reader on to the next row of holes.	CI

Each pair of Function Letters, as used in an Assembler program, may be written in any order; the assembler will then compile the pair into the correct bit-pattern at machine-code level as shown above. In hardware terms, the right-hand three F bits mainly affect the serial data-paths leading *out of* the accumulator; the left-hand three F bits mainly affect the way the *inputs* to the ALU are logically combined on their way into to the accumulator. In reference 5, certain Assembler letters are termed *Auxiliary Function letters;* these are X, N, D, H and C.

As reference 5 observes, "certain other pairs of Function letters [ie Assembler nmemonics] are permissible though of limited utility". Amongst these are the following:

010110	a' = a(x + 2a)	MD
011110	a' = a(x + a/2)	MH

It is also possible to use an unconditional jump (T) or a conditional jump (Z) in combination with the *Auxiliary Function letters* (X, N, D, H, C) to achieve some curious side-effects concerned with either (a) inhibiting the normal incrementing of the program counter register PC, or (b) forcing the logical effect of the *Auxiliary Function letter* to persist until the next unconditional or conditional jump instruction is encountered. Examples are:

F bits	***action***	***inhibit +1 to PC?***	***Auxiliary Fn persists?***	***corresponding Assembler letters***
100011	a' = a	yes	no	TX
101011	a' = - a	yes	no	TN
010011	a' = 2a	no	yes	TD
011011	a' = a/2	no	yes	TH
001011	a' = 0	no	yes	TC

As reference 5 points out, “Great care is needed when using such combinations and it is recommended that their use be avoided until some experience has been gained”. The TX and TN orders are always to be placed in the second (more significant) half of the word containing a pair of instructions. At the conclusion of a sequence of pairs of instructions, the second half of each of which contains TX or TN, control will be returned to the order in the memory location following that which contains the first instruction in the sequence. Such sequences are used in the multiplication and printing subroutines (see section E1/X4). When using the TD, TH and TC instructions, the contents of the accumulator will be doubled, halved or cleared respectively *every* time PC is incremented until a successful jump instruction is encountered.

The Function letter X is only intended for use with the Function letters T and Z, as described above. An instruction with X alone (ie with F-bits = 100000) has no effect. The Function letter Y is used in the multiplication subroutine to manipulate the accumulator extension register (see below). Multiplication (M) and printing (P) are performed by special subroutines. M and P orders must be placed in the second-half (ms half) of a 32-bit program-word.

Nicholas’ speed of operation is given as approximately 80 instructions per second (reference 7) or 100 instructions per second (reference 6); the add-time is quoted as 12.5 milliseconds on average (reference 8).