Instruction sets and instruction times for the Elliott 4100 Series computers.

General considerations.

The Elliott 4100 series' 24-bit word employed two's complement representation for integers. Single-address format instructions were either short (12 bits) or long (24 bits). In the context of the early 1960s, the instruction set contained a rich variety of orders with due consideration being given to the needs of compiler writers for block-structured languages such as Algol.

Short instructions have six function-bits (ie the op code) and six address bits, thus only giving access to the first 64 memory locations for operands.

Short format instructions:

6	6
F	**N**
Op code	Address

For short instructions, the octal value of the F-bits lies in the range 00 to 37. Almost all of the 32 short instructions are duplicated by long instructions that offer richer addressing modes. Thus, short instruction op codes 00 to 27 generally have the same arithmetical or logical definitions as long instructions that have op codes in the range 40 to 67 – (see the lists given below).

The layout of long, 24-bit, instructions is as follows:

Long format instructions:

6	2	1	15
F	**Y**	**Z**	**N**
Op code	Addressing mode	Extracode indicator	Literal or address

For long instructions, the octal value of the F-bits lies in the range 40 to 77. The values of the Y-bits denote the addressing mode, as follows:

0:	literal	(N is treated as a positive integer – ie, not signed)
1:	direct	(N is an absolute address)
2:	modified	(the contents of R is added to N to give the final address)
3:	indirect	(the contents of memory location N gives the address of the operand).

When using modified or indirect addressing, bits 16 – 22 of the final address must be zero. This gives the effect of being able to address a maximum of 256K words, in blocks of 64K words.

The Elliott 4100 series has the following programmer-accessible registers (see below). When describing digit-positions, the 4100 convention is that bit 24 is the left-hand (most-significant) position and bit 1 is the least-significant position.

	size, bits	***description***
M	24	main accumulator
R	24	reserve accumulator, also used as the address-modification register, etc.
S	17	program-counter, also known as the sequence-control register
K	12	count register
C	14	conditions register. Bits 16 – 7 are unallocated. The remaining C bits are assigned as follows (according to an amalgamation of the information contained in [1 and 8]:

c24: result negative, denoted by *Neg* in the instruction listing given below
c23: result standardized, denoted by *St*
c22: result non-zero, denoted by *Nz*
c21: carry-out from ms accumulator bit during addition or subtraction, denoted by *Ca*
c20: arithmetic overflow, denoted by *Of*
c19: normal interrupt permit
c18: attention interrupt permit
c17: invalid information transfer
c6 – c1: these give the state of six manual switches on the operator's console.

The C bits can be inspected *in toto* by transferring C into the accumulator, M, by the 700/520 instruction (see below). In some instances, a 24-bit link is formed by adding the current value of the program counter S to bits 24 – 18 of C, thereby preserving the essential control-state of a program in a compact form prior to entering a subroutine.

Elliott 4100 series instruction set.
In the instructions listed below, m is the contents of M, r is the contents of R, s is the contents of S, and k is the contents of K. Primes indicate the new values at the conclusion of an instruction, a notation chosen in other sections of the *Our Computer Heritage* website. It is significant that, in the original Elliott-Automation documentation for the 4100 series, the Algol symbol for 'becomes equal to' (:=) was used instead of primes, a hint that the architecture of the 4100 series computers was oriented towards the needs of high-level language compiler writers. In the list below, square brackets indicate 'contents of address'; thus, [r] means 'the contents of location addressed by the R register'. There are four six-bit characters per word; these are denoted by {a, b, c, d}, where a is stored at the most significant end of the word.

There are several choices of operand-addressing for each combination of the F bits (the op code), depending upon how the Y bits are to be interpreted. Rather than show every option explicitly in the listing below, we show the action for short instructions and then the action for long instructions. For the latter, three possibilities are distinguished in the listing below:

(a) the action is independent of the setting of the Y bits;
(b) a specific action is defined for the combination Y = 0;
(c) a specific action is defined for the cases of Y = 1, 2 or 3.

In the listing below, these three cases are distinguished by filling in the Y column by: (a) nothing, (b) y = 0; (c) y = 1,3. Finally, the actions for the shift instructions, the register-to-register instructions, the input/output instructions and the extracode instructions also depend upon the values of the N digits. For clarity, these four sub-groups of instructions are listed separately, after the straightforward computational orders have been described. Since the Z field in an instruction is zero for all except extracodes, the Z field has been omitted from all but one of the sub-sections below. The values of the F-bits are given in octal below.

The behind-the-scenes hardware decoding of the F, Y, Z and N fields of a long instruction is complex. For this reason, the documentation issued to Elliott users laid emphasis on assembly-language mnemonics, rather than on bit-patterns, when tabulating instructions. The mnemonics, reproduced below, are those used in the NEAT (National Elliott Assembly Technique) and the SAP (Symbolic Assembly Programming language) programming manuals.

Group A: straightforward short and long instructions.

F (short)	***F (long)***	***Y***	***principal action(s)***	***description***	***mnemonic***
00	40		m' = m + Q	Add, using main accumulator	ADD
01	41		m' = m – Q	Subtract	SUB
02	42		m' = Q – m	Reverse-subtract	NADD
03	43		m' = Q	Load accumulator	LD
04	44		r' = Q	Load reserve accumulator	LDR
05			s' = n; $c'_{24\text{-}18} = n_{24\text{-}18}$	Exit	JIR
	45	y = 0	s' = N	Unconditional jump	J
	45	y = 1,3	s' = Q	Unconditional jump	JI
06	46		m' = m & Q	Logical AND	AND
07	47		m' = m & ⌐Q	Logical AND NOT	ANDN
10	50		r' = r + Q	Add, using reserve accumulator	ADDR
11	51		r' = r – Q	Subtract	SUBR
12	52		r' = Q – r	Reverse subtract	NADR
	53	y = 0	0' = $_{c24\text{-}18}$ + s; s' = s + N	Subroutine entry, addr zero for link	JFL
13	53	y = 1-3	0' = $_{c24\text{-}18}$ + s; s' = Q	Subroutine entry, addr zero for link	JIL
14	54		k' = Q	Load K, the count register	LDK
15			*shift instructions – see explanation below*		
	55		compare (m – Q)	set the conditions register accordingly	COMP
16	56	y = 0	s' = s + N	unconditional relative jump forwards	JF
	56	y = 1,3	s' = s + Q	unconditional relative jump forwards	JA
17	57	y = 0	s' = s – N	unconditional relative jump backwards	JB
	57	y = 1,3	s' = s – Q	unconditional relative jump backwards	JS
20	60	y = 0	if *Neg* then s' = s + Q	relative jump forwards if negative	JN
21	61	y = 0	if not *Neg* then s' = s + Q	relative jump forwards if not negative	JNN
22	62	y = 0	if not *Nz* then s' = s + Q	relative jump forwards if zero	JZ
23	63	y = 0	if *Nz* then s' = s + Q	relative jump forwards if non-zero	JNZ
24	64	y = 0	if *St* then s' = s + Q	relative jump forwards if standardized	JST
25	65	y = 0	if *Of* then s' = s + Q	relative jump forwards if overflow	JOF
26			(unassigned?)		
27	67	y = 0	k' = k – 1; if k_{12} = 1 then s' = s + Q	decrement, test and jump if	DKJN
30	60	y = 1,3	Q' = m	store accumulator	ST

31	61 y = 1,3	Q' = r	store reserve accumulator	STR
32	62 y = 1,3	Q' = – Q	negate the contents of memory	NEGS
33	63 y = 1,3	Q' = Q – m	subtract acc from store	SUBS
34	64 y = 1,3	Q' = Q + m	add acc to store	ADDS
35	65 y = 1,3	Q' = 0	clear memory location	CLS
36	66 y = 1,3	Q' = Q + 1	increment memory location	INCS
37	67 y = 1,3	Q' = Q – 1	decrement memory location	DECS
	70 y = 0	*register-to-register moves – see explanation below*		
	70 y = 1,3	Q' = Q(bcda); m' = m(abc)Q(a)	fetch next character	GET
	71 y = 1,3	Q' = Q(bcd)m(d)	store next character	PUT
	72 y = 1,3	m' = (r,m)/Q	divide, double-length	DIVM
	73 y = 1,3	(r,m)' = (r,m) x Q	multiply, double-length	MULM
	74 y = 1,3	m' = Q; [r]' = m; r' = r – 1	pop up from a stack	MVE
	75 y = 1,3	Q' = m; m' = [r]; r' = r + 1	push down onto a stack	MVB
	76 y = 1,3	swap Q and m	exchange values of Q and m	EXC
	77 y = 1,3	swap Q and r	exchange values of Q and r	EXCR

The input/output instructions, for which the F bits = octal 74 to 77 and for which the Y bits = 0, are described later.

Group B: shift instructions.
The shift instructions, which are short orders for which the F bits = octal 15, use the N bits to determine the mode of shifting (ie left or right, logical, arithmetic or circular) and the K bits to determine the number of places shifted. The list of permitted possibilities is as follows, in which the value of the six N digits is given in octal:

F	***N***	***action***	***mnemonic***
15	00	shift r left arithmetically k places	SRL
15	01	shift r left circularly k places	SRLA
15	02	shift r right arithmetically k places	SRR
15	03	shift r by k 6-bit characters circularly left	SRLC
15	04	shift m left arithmetically k places	SML
15	05	shift m left circularly k places	SMLA
15	06	shift m right arithmetically k places	SMR
15	07	shift m by k 6-bit characters circularly left	SMLC
15	12	shift r right logically by k places	SRRL
15	16	shift m right logically by k places	SMRL
15	20	shift r until standardized, or k places, whichever is less	SRST
15	24	shift m until standardized, or k places, whichever is less	SMST
15	40	shift both m and r arithmetically left k places	SBL
15	42	shift both m and r arithmetically right k places	SBR
15	52	shift both m and r logically right k places	SBRL
15	62	shift m and r until standardized, or k places, whichever is less	SBST

Group C: register-to-register instructions.
The register-to-register instructions, which are long orders for which the F bits = octal 70 and the Y bits = 0, use the N bits to define the registers involved. The list of assigned combinations, for which the value of the 15 N digits is given in octal, is as follows:

F	*Y*	*N*	*action*	*mnemonic*
70	0	00020	r' = k	KTOR
70	0	00402	r' = m	MTOR
70	0	00404	r' = s	STOR
70	0	00441	r' = r + 1 if carry set	CAIR
70	0	00541	r' = r – 1 if carry set	CADR
70	0	01001	m' = r	RTOM
70	0	01003	m' = m OR r	MORR
70	0	01010	m' = c	CTOM
70	0	02001	s' = r	RTOS
70	0	02002	s' = m	MTOS
70	0	04002	c' = m	MTOC
70	0	10001	k' = r	RTOK
70	0	10002	k' = m	MTOK
70	0	10201	k' = – r	RNTK
70	0	21000	m' = interrupt word (see below)	ITOM
70	0	41000	m' = attention word (see below)	ATOM

Group D: input/output instructions.
The input/output instructions, for which the F bits = octal 74 to 77 and for which the Y bits = 0, use the first three octal digits of N to supplement the F bits. The last two octal digits of N, denoted as *nn* below, define the peripheral channel number. The 4100 Standard Interface normally provides for up to 12 independent, asynchronous, input/output channels – (with extra channels as an option, up to 14?). Each channel can call for either of two types of program break: an *Interrupt* or an *Attention*. Two 12-bit locations, the Interrupt word and the Attention word, are provided and each may be inspected by program using the ITOM and ATOM instructions above. Beneath this level, a hardware *Hesitation* (high-priority interrupt) is also provided for use with devices using hardware-assisted autonomous data transfers (ADT) and cycle-stealing.

A hardware Autonomous Transfer Unit (optional for the 4120, built-in for the 4130) organizes bulk data transfers via cycle-stealing in a manner independently from the main CPU. Thus, input/output activity could be interleaved with normal computing. Up to three *packed transfer units* and one *unpacked transfer unit* may be included in an Autonomous Transfer Unit.

The 4100 Standard Interface has eight *data-in* lines, 8 *data-out* lines, three interrupt lines and eleven other control, status and timing signals. The input/output instructions for peripheral channel nn are as follows:

F	*Y*	*N*	*action*	*mnemonic*
74	0	000nn	Input data packed repetitive	IDPR
74	0	100nn	Output data packed repetitive	ODPR
74	0	200nn	Input data unpacked repetitive	IDUR
74	0	300nn	Output data unpacked repetitive	ODUR
75	0	000nn	Input status word packed repetitive	ISPR
75	0	100nn	Output control word packed repetitive	OCPR
75	0	200nn	Input status word unpacked repetitive	ISUR
75	0	300nn	Output control word unpacked repetitive	OCUR

76	0	200nn	Input data unpacked single to m	IDUM
76	0	300nn	Output data unpacked single from m	ODUM
77	0	200nn	Input status word unpacked single to m	ISUM
77	0	300nn	Output control word unpacked single from m	OCUM

Group E: Extracodes.
When Z = 1 and the F-bits are in the octal range 40 to 77, an extracode instruction may be indicated, though only about 26 of the available F-bit combinations are allocated as actual extracodes. The action upon encountering an extracode instruction varies according to whether the Y bit (ie the address-mode bits) are zero or in the range 1 to 3, as follows:

Action for literal address mode (Y = 0):
(a) place N in memory location 1;
(b) place the link (c_{24-18} + S) in memory location 2;
(c) jump to a memory location given by twice the value of the F-bits, ie to one of the even-numbered locations in the range 64 to 126 inclusive. This is then the start of a standard subroutine for implementing the extracode.

Action for other addressing modes (Y = 1, 2 or 3):
(a) if Y = 1 then place N in location 1, or
if Y = 2 then place (N + r) in location 1, or
if Y = 3 then place the contents of address N in location 1;
(b) place (c_{24-18} + S) in memory location 2;
(c) jump to a memory location given by twice the value of the F-bits plus 1, ie to one of the odd-numbered locations in the range 65 to 127 inclusive.

The extracodes are now listed, with F, Y, Z and N being given in octal except that, for the Y field, a 'y' indicates any number in the range 1, 2 or 3 and for the N field, an 'n' indicates any valid address. Note that the 14 floating-point extracodes are implemented in hardware on the Elliott 4130. On the 4130 where hardware is used, the mantissa occupies 48 bits within CPU registers and the exponent 12 bits. This *triple* can be accessed collectively via the WUF and FLU extracodes (see below). When held in memory, floating-point numbers are normally rounded and packed into two words containing 39 bits of mantissa and 9 bits of exponent. The following abbreviations are used in the list below:
fpa = floating-point accumulator;
fQ = the floating-point operand held in locations Q, Q+1.
dQ = the double-length operand held in locations Q, Q+1.
tQ = the triple-length operand held in locations Q, Q+1 and Q+2.

F	***Y***	***Z***	***N***	***action***	***mnemonic***
40	0	1	0	fpa' = – fpa	FN
40	0	1	2	fpa' = integer m in floating-point form	FCP
40	0	1	4	fpa' = modulus (fpa)	FMOD
40	0	1	6	m' = entier (fpa)	FENT
41	0	1	10	if fpa < 0, m' = – 1; if fpa = 0, m' = 0; if fpa > 0, m' = 1	FSIG
40	y	1	n	fpa' = fQ	FL
41	0	1	0	copy to lower address	CTLA
41	0	1	1000	copy to higher address	CTHA
41	y	1	n	fQ' = fpa	WF
42	y	1	n	fpa' = fpa + fQ	FA

43	y	1	n	fpa’ = fpa – fQ	FS
44	y	1	n	fpa’ = fpa x fQ	FM
45	y	1	n	fpa’ = fpa / fQ	FD
46	y	1	n	set c_{24-22} from (fpa – fQ)	FCP
50	y	1	n	m’ = m x Q	MULS
51	y	1	n	m’ = m / Q; r’ = remainder	DIV
52	y	1	n	(r,m)’ = Dq	BL
53	y	1	n	dQ’ = (r,m)	WB
54	y	1	n	jump indirect and restore link	JIRX
55	y	1	n	jump indirect	JIX
56	y	1	n	jump indirect, setting link	JILX
57	y	1	n	access chapter item with index Q, placing its addr in R	INDEX
60	y	1	n	fpa’ = tQ (unrounded representation)	FLU
61	y	1	n	tQ’ = fpa (unrounded representation)	WUF
77	0	1	n	n^{th} letter of alphabet displayed (on console)	TR
77	y	1	n	Q displayed in octal (on console)	CH

The INDEX instruction in the above list assumes that a program’s memory-space is organised into *chapters* – see also section E6/X4 – and that a particular chapter contains some form of structured data such as an array or table. If an INDEX instruction is issued with the address of a codeword (or descriptor) in R, then the address of the i^{th} element of the data-structure to which the codeword points is placed in R.