Appendix 7
Technical Details of the Elliott 4100 Series Computers

A7.1 General Overview of the Family

The specification and hardware design for the Elliott 4100 series had largely been completed by the start of 1964 [1]. Alan Wakefield, an Elliott Sales engineer, recalls [2] that 'we, the Sales force, were introduced to the 4100 series, internally, at a three-day sales training course on December 14th, 15th and 16th 1964. Customer visits, demonstrations, seminars and sales campaigns started in earnest early in 1965.' The Elliott 4100's instruction set represented a new direction for Borehamwood, in that it strongly reflected the influence of programmers such as Roger Cook (see also Chap. 8). Roger Cook remembers [3] that: 'we designed the 4120 with NCR in mind and the 4130 for our more technical users. We were also dimly aware that compilers would become the norm.' Evidence of the hardware support for high-level languages is given in Sects. A7.2 and A7.3.

There were two members of the 4100 series family, of which the 4120 was the first to be delivered to an outside customer in 1965. By 31 March 1967, 58 of the 4120 systems (including the ARCH 2020 variant) had been delivered and a further 86 machines were on order [4]. The more powerful Elliott 4130 computer was the next to arrive. By 31 March 1967, 21 of the 4130 systems (including the ARCH 2030 variant) had been ordered but none had yet been delivered to an external customer.

When Elliott-Automation's main computing interests were absorbed into ICL at the end of 1968, the 4100 series were the only Elliott computers to be marketed by ICL, albeit as a poor relation to the more comprehensive and powerful English Electric System 4 and ICT 1900 series machines and to the last of the LEO III computers (LEO III deliveries spanned the period 1962–1969). Many of the surviving 4100 series software technical manuals are held in the ICL Archive [5]. As mentioned in Chap. 13, ICL did at least acknowledge in 1968 that the Elliott 4100 series had been 'most successful in universities, research establishments and for industrial automation schemes. The graphical display system associated with 4100 central processors is a recognized leader in computer design technology' [6].

The Elliott type 4280 Graphical Display Unit had a 10 in. × 10 in. viewing area with 1,024 × 1,024 addressable positions, giving a resolution to within 0.01 of an inch. The image was automatically re-generated ten times per second from a display file held in the computer's main memory, to give a flicker-free picture. Curves or lines could be drawn at the rate of 100 μs per displayed inch. The type 4280 unit included a vector generator for drawing straight lines, a character generator which displayed the alphanumeric characters of the 4100 series internal codes in three sizes (5/64th in., 5/32nd in. or 5/16th in.) and a hand-held *light pen* which enabled the user to identify points on the display. (The light pen achieved the same effect as a modern mouse and cursor.)

Despite its apparent enthusiasm for Elliott's graphical display system, ICL showed little interest in maintaining the existing 4100 machines or their peripherals. Largely because of this, three ex-Elliott-Automation engineers founded Systems Reliability Ltd. (SRL) at Luton in 1968, as described in Chap. 14.

According to [7] manufacture of the 4100 series ceased in 1970. It is not known when the last 4100 was retired from active service but it is likely to have been at the beginning of the 1980s.

The 4100 series machines were based on bit-parallel CPUs employing silicon transistors. For each of the 4120 and 4130 computers, there was originally a choice of ferrite core store: either 2- or 6-μs cycle time. The 4130 had a faster CPU and implemented floating-point operations in hardware. Floating-point was performed by software extracodes in the 4120. The Autonomous Transfer Unit (ATU) was an add-on extra for the 4120 but was built into the main cabinet of a 4130. The comparative figures given in Table A7.1 are taken from [8], which is dated October 1967.

The physical dimensions of a basic Elliott 4130 processor were: height 63.5 in. (161 cm), width 70 in. (178 cm), depth 26 in. (68.5 cm), weight 1,040 lb (450 kg) and power consumption 2 kVA.

Table A7.1 Overall characteristics of Elliott 4100 series computers

	4120	4130
Word length, bits	24	24
Number of instructions per word	1 or 2	1 or 2
Max. installed memory at 2 μs cycle time	64K words	256K words
Fxpt ADD time, direct addressing mode, with 6 μs store	12.0 μs	12.0 μs
Fxpt ADD time, direct addressing mode, with 2 μs store	5.6 μs	4.5 μs
Fxpt MPY time, direct addressing mode, with 6 μs store	67.0 μs	22.0 μs
Fxpt MPY time, direct addressing mode, with 2 μs store	60.6 μs	15.0 μs
Floating-point hardware?	No	Yes
Flpt ADD time, direct addressing mode, with 2 μs store	199 μs	15 μs

A7.2 Systems Architecture and Visible Registers

The Elliott 4100 series' 24-bit word employed two's complement representation for integers, as did all Elliott computers. Single-address format instructions were either short (12 bits) or long (24 bits). A long instruction may be split across two consecutive memory locations, in which case the instruction takes a little longer to execute. The program counter, S, refers to the half-word address of the next instruction. The effect of obeying an instruction in the second half of a word which has just been altered by the instruction in the first half of the same word is not defined. For many applications, four 6-bit characters were stored per word.

Roger Cook has commented retrospectively [3] that: 'I (wrongly?) decided that a 6 bit (rather than 8 bit) character would be cost effective – having been told that memory was "quite expensive"'. When held in memory, floating-point numbers were normally rounded and packed into two words containing 39 bits of mantissa and 9 bits of exponent (see also Sect. A7.3).

As is described later, the Elliott 4100 series had a much more powerful repertoire of addressing modes than previous Borehamwood computers. The series also implemented a neat and comprehensive mechanism for handling the various condition-bits (e.g. accumulator sign, overflow, interrupt, etc.) that contribute to the current status of a computer. Roger Cook has observed [3] that: 'As much as possible of the internal operation was to be available to the programmer (eg interrupts, busy etc) and was to be mathematically consistent'. The concept of a stack, as used for example in block-structured languages, is given hardware support – see the MVE and MVB instructions given in Sect. A7.3. There was hardware assistance for packing and unpacking characters. A distinction was made between the three forms of shift, namely *logical*, *arithmetic* or *circular*. There was a flexible set of instructions for register-to-register moves. Finally, the Elliott 4100 series had a Standard Interface for all input/output devices, the physical manifestation of which was a standard plug and socket arrangement for all peripherals.

Short instructions for the Elliott 4100 series have six function-bits (i.e. 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 00–37. Almost all of the 32 short instructions are duplicated by long instructions that offer richer addressing modes. Thus, short instruction op codes 00–27 generally have the same arithmetical or logical definitions as long instructions that have op codes in the range 40–67 (see lists in Sect. A7.3).

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 40–77. The values of the Y-bits denote the addressing mode, as follows:

0	Literal (N is treated as a positive integer – i.e. 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 has the following programmer-accessible registers. 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, 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.

A7.3 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 this book to be compatible with the other instruction sets described in Appendices 2–6. It is significant that, in the original Elliott-Automation documentation for the 4100 series [1, 8], the Algol symbol for 'becomes equal to' (:=) was used instead of primes, yet another 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 6-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 showing 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; and (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 users (e.g. [8]) 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. Roger Cook remembers [3] that: 'The symbolic names for the instruction codes were specified by NCR – (we [at Borehamwood] tended to program using the basic numerical codes).'

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

(continued)

(continued)

F (short)	F (long)	Y	Principal action(s)	Description	Mnemonic
03	43		$m' = Q$	Load accumulator	LD
04	44		$r' = Q$	Load reserve accumulator	LDR
05			$s' = n$; $c'_{24-18} = n_{24-18}$	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		$m' = m \,\&\, \neg Q$	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	$y = 0$	$0' = {}_{c24-18} + s$; $s' = s + N$	Subroutine entry, addr zero for link	JFL
13	53	$y = 1\text{-}3$	$0' = {}_{c24\text{-}18} + s$; $s' = Q$	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

(continued)

(continued)

F (short)	F (long)	Y	Principal action(s)	Description	Mnemonic
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 74–77 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 (i.e. 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 74–77 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.

An optional hardware Autonomous Transfer Unit 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 8 *data-in* lines, 8 *data-out* lines, 3 interrupt lines and 11 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

(continued)

(continued)

F	Y	N	Action	Mnemonic
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 40–77, 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, as deduced from [1], varies according to whether the Y bit (i.e. the address-mode bits) are zero or in the range 1–3, 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, that is to one of the even-numbered locations in the range 64–126 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, that is to one of the odd-numbered locations in the range 65–127 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	1,000	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 × 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 × 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	(Unrounded representation)	WUF
77	0	1	n	nth 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 under SPAN in the next section – 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 ith element of the data-structure to which the codeword points is placed in R.

A7.4 Operating Systems, Time-Sharing and Multiprogramming Facilities

According to [9], there was originally an intention to provide hardware assistance for multiprogramming. The 4130 was intended to offer two programming environments: *Executive Mode* and *Protected Mode.* Under the latter, which was to be the normal user mode when the multiprogramming system was in operation, a program was restricted to its allocated area of memory and was limited in its use of peripheral devices. An *Alarm Clock* was to be provided that set a limit on the time for which a particular user program could run within Protected Mode before being terminated. Within Protected Mode, core store was to be allocated to a user program via two 10-bit registers that gave the *Base* address and the *Range* of permitted memory. Any attempt to access a location outside the permitted area would cause the user's program to be suspended and Executive Mode entered.

To facilitate the above proposed multiprogramming environment, several additional Elliott 4130 instructions were suggested, including:

EXEN	Enter Executive Mode
PMEN	Load the Base and Range registers and the Alarm Clock setting and then enter Protected Mode

It is believed that the above multiprogramming facilities were seldom activated for the Elliott 4130 computer, except in the KOS operating system described later. An *Alarm Clock*, actually known as the *Real Time Clock*, was, however, provided as standard for the 4130 and as part of the Autonomous Transfer Unit for the 4120. This produced an interrupt once every second and could be set to 'ring' by transferring data after N seconds.

From [10] it is deduced that the default operating system for Elliott 4100 computers in the period from 1965 to 1968 was the Systems Executive known as EASE. This consisted of three sections: NICE (Normal Input and Control Executive), SPAN (Storage Planning and Allocation) and TSS (Time Sharing Supervisor). It is believed that the three sections of EASE constituted independent modules, as follows.

NICE was a simple Executive that enabled an operator to input relocatable binary paper tapes, enter a named program, remove a program, cause a printout of an area of memory, etc.

SPAN handled the housekeeping for information transfers between primary memory (core) and secondary memory (e.g. disc pack). It assumed that a program's storage space was divided into *chapters*, each containing one or more *blocks*, each block containing one or more entry points known as *labels*. A program may call upon one or more of SPAN's routines, which include the following utilities:

ALLOC	asks SPAN to reserve space (in primary memory) for a chapter. This may result in the response *No Room* if SPAN cannot find sufficient space
DELETE	frees up space no longer needed by a chapter
BANISH	move a chapter to secondary storage
RECALL	bring a chapter into primary storage

TSS was available to look after *Interrupts* and *Attentions* coming from each Standard Interface channel, transferring control to a routine appropriate for each peripheral device. The Elliott 4100 series defines three levels of program priority, the highest being called the Interrupt level, the intermediate one being called the Attention level and the lowest level being that of normal computation. There exist appropriate TSS routines running at each level. Transfers to/from a level within TSS occur either as a result of an Interrupt or Attention signal or as a result of subroutine entry/exit. TSS organises queues and buffers as appropriate, for input/output transfers.

An early in-house Elliott 4100 operating system used on the commissioning floor at Borehamwood was called SysD (System D). This offered 'a very simple system control vocabulary, was interactive on the Selectric typewriter, and made it easy to load and run paper tape based programs' [11].

By the 1970s, it is thought that several other operating systems had been implemented for Elliott 4100 series computers. DES (Disc Executive System) was a standalone operating system used by individual large users, for example when working overnight and at weekends. It allowed programs to access the whole of physical memory. DES2 had a 'slave' area of memory where a second program could be run in tandem with the first program. DES BATCH was the batch job operating system, used for example in computing service environments.

DES reflected the fact that the Elliott 4100 series was amongst the first Borehamwood products to offer disc packs as standard peripherals. Disc (or *disk*) memory – in fixed or exchangeable form – was one of the significant innovations of the mid-1960s. Disc storage as an intermediate, *direct-access,* medium between core primary memory and magnetic tape backing store had begun to make its presence felt in the late 1950s, following the trend set by IBM in 1956 with their IBM 305 RAMAC product – as mentioned in Chap. 10. However, a few years were to elapse before disc technology made a significant impact upon the design of standard operating systems. The disc equipment offered for the Elliott 4100 series computers in 1967 consisted of ten disc surfaces, each with 100 tracks split into 16 sectors of 64 words. The on-line capacity was therefore about one million words. The read-write heads took about 100 ms to traverse 33 tracks and, once the desired *cylinder* of tracks had been reached, the mean access-time was 12.5 ms and a sector could be transferred in 1.5 ms [8].

Elliott 4100 series computers were installed at several UK Universities, where they inspired systems software developments by the academic users (see also Sect. 8.5). The University of Kent at Canterbury was especially active, being responsible for the Kent On-line System KOS [12], a simple multi-access operating system which allowed both batch use and on-line terminals simultaneously. KOS, implemented on an Elliott 4130 in the period late 1968 to early 1970, supported a fully conversational incremental BASIC compiler via eight teletype terminals [13]. The development of the BASIC compiler was a joint venture between Kent and the University College of North Wales.

Another, unrelated, on-line multi-access system had come live in 1967 when the functional language POP-2 and Multi-POP was implemented on an Elliott 4120 at the Department of Machine Intelligence and Perception at Edinburgh University [14]. As further evidence of the interest amongst researchers in the Elliott 4130's features, the Institut de Programmation at Orsay (south of Paris) of the Institut Blaise Pacal ordered a 4130 in 1967 'pour le *time sharing et le conversationnel*'. To quote [15], 'We are about to receive an Elliott 4130, equipped for time-sharing use, and possessing a wired structure which will enable us to develop advanced studies on programming systems, memory hierarchy management, handling of virtual memories.'

Returning to KOS, it is worth quoting Peter Brown, the leader of the KOS development team at Kent, because of his appreciation of some of the Elliott 4130's more advanced features [12].

'The 4130 has two modes of operation, namely *executive mode* and *slave mode.* The latter, however, has been completely ignored until very recently and virtually all programs, whether written by the manufacturers or by users, have run in executive mode. The slave mode of operation uses the well-known base and range register technique. There is an important sub-mode of slave mode, called *pure procedure mode.* (The manufacturers call it *common program mode.)* This has influenced the design of KOS more than any other hardware feature. In pure procedure mode the program instruction counter is absolute and it is only data references that are subject to the addition of the base. Using this hardware, a single program can control any number of slaves. A slave is simply an area of workspace, and the slave that is active at any one time is determined by the setting of the base and range registers.

'There is a further hardware feature that allows for pure constants. Each pure procedure can have its own data area which contains all true constants, i.e. all data whose value is set at load time and not subsequently changed. When using pure procedures, it is best to place tables and error messages in this data area, so that there will only be a single copy of them rather than one copy in each slave area.

'The pure procedure mode of the hardware and its ease of use radically changes the economics of time sharing for the 4130. For multi-access work the 4130 is at its most efficient when there are one or more pure procedures, remaining permanently in store while they are in use, being shared by several jobs. For example, there might be two pure procedures in use, a compiler and an application package; three console users and a card-to-printer job might be using the compiler, and four other console users might be using the application package. All should enjoy excellent response time. Moreover, if the pure procedures are always resident in core there is no overhead in providing fully conversational working.'

KOS allowed both the Algol and the Fortran compilers to be resident in the batch stream [16]. Brian Spratt, at the time the Director of the Computing Service at the University of Kent, remembers [16] that KOS proved very successful and was adopted by several of the other UK universities that used Elliott 4130 computers in their Computing Services for running student's programs. 'The Computer Board [which administered UK academic computing resources] was quite pleased and we (ie Kent and nine other universities) all had 64K 2 microsecond store upgrades approved. But our (Ex Elliott) computer manager Des Caul managed to upgrade the system with the original 6 microsecond store to give us a total of 96K memory. The other 4130 sites quickly adopted this arrangement.' In all, Elliott 4100 series machines were delivered to about 18 UK universities and about five technical colleges – see the delivery lists in Appendix 8.

Peter Brown ends his description of KOS by comparing KOS with the Elliott/ICL software currently available in 1971 for the 4130 and states [12] that 'the manufacturers have recently introduced a new multi-programming operating system. Currently this is little used but it will, no doubt, gradually find wider usage'.

In contrast to the enthusiasm at Kent, Leicester University never used KOS on its large 4130 installation consisting of 65K words of 2-μs core store, an Autonomous Transfer Unit and eight magnetic tape decks. Leicester's original operating system was T30C, essentially a batch processor. John Thompson, who ran the Leicester Computing Centre, has commented thus [17]. 'Algol fitted the Elliott software environment perfectly, but the Fortran team were adamant that one could not have a Fortran system in that framework and designed their own from scratch. So T30C essentially had to dump one system and spool in another in order to change language. For speed, we allocated three tape drives to the system because we could afford to. The Fortran compiler was pretty awful (we were finding bugs for ages and Elliott were slow to fix them) and certainly one could not access the top half of the store. (Whether one could in Algol I cannot remember, but most of the big linear algebra calculations were in Fortran, as were imported programs such as IBMOL.) When we finally got hold of the Fortran compiler details - by which I mean a poorly annotated listing - I was able to jury rig the thing so that a particular named COMMON block, /ZZZZ/ actually, resided in the top half of the store.'

References

1. Anon (1964) Elliott 4150 digital computer: functional specification. 44-page typed internal report TIS/4150, published by Scientific Computing Division, Elliott Brothers Ltd., London, dated May 1964. There are hand-written alterations, amending the 4150 to 4120
2. A Wakefield (2009) E-mail to Simon Lavington, 10 Jan 2009
3. Roger Cook (2008) E-mail to Simon Lavington dated 11 Oct 2008
4. Elliott automation: computers and orders, 1947–1966. Booklet published in early 1967 by the Directorate of Information Services, Elliott-Automation Ltd., 21 Portland Place, London W1. Addendum sheet gives the totals at 31 March 1967
5. The ICL Archive, preserved by the National Museum of Science and Industry (London) at its repository at Wroughton, has been catalogued by Hamish Carmichael of the Computer Conservation Society. Elliott 4100 series technical manuals, mostly concerning software, are held as the following ICL Archive item numbers: 38/152 to 38/155, 38/163, 38/181 to 38/201, 38/215
6. Birkett D (1968) ICL Profile, a six-page Press Release issued by the ICT Company Information Officer, Dennis Birkett, dated 10 Aug 1968 (Available as document COM/1993/1433 at the National Museum of Science and Industry, South Kensington, London)
7. Campbell-Kelly M (1989) ICL – a business and technical history. Oxford University Press, Oxford. ISBN 0-19-853918-5
8. Anon (1967) 4100 computer system FACTS booklet, October 1967. Published by Elliott-Automation Computers Ltd. Catalogue number 440 – issue 4
9. (a) Anon, Appendix C: NCR – Elliott 4100 Data processing system: 4130 processor and (b) Anon, Appendix E: 4100 Autonomous Transfer Unit, Functional Specification, Section 2. These two typed appendices were evidently part of a larger internal document produced by Elliott-Automation. Undated but probably written between mid-1965 and October 1967. Documents preserved by D H (Dennis) Rowland, who worked for the Airspace Control Division of Elliot Brothers Ltd., London
10. Anon (1965) NCR Elliott 4100 Electronic Data Processing System FACTS, Apr 1965. Jointly published by (a) Scientific Computing Division, Elliott Brothers Ltd., Borehamwood, London and (b) NCR Electronics, National Cash Register Company Ltd., London. Catalogue number 440
11. David Warman (2010) E-mail to Simon Lavington dated 3 Feb 2010
12. Brown PJ (July 1971) The Kent on-line system. Softw Pract Experience 1(3):269–277
13. Brown PJ (1971) Effective use of batch and on-line facilities for introductory programming courses. Comput Bull Oct:370–372
14. Burstall RM, Collins JS, Popplestone RJ (1971) Programming in POP-2. Edinburgh University Press, Edinburgh
15. Arsac J, A Propos de la Reforme de l'Enseignement. In Progrès et Science, special issue on the Institut de Programmation, 4th trimester, 1967, page 6. Background comments on this article were provided by Pierre Mounier, in an e-mail to Simon Lavington dated 14 Apr 2009. See also the cartoon, reproduced as Fig. 8.9 in Chap. 8, which was drawn by Bernard Robinet, a student at the Institut de Programmation who was writing his dissertation on APL in 1967
16. EB Spratt (2008) E-mail to Simon Lavington, 29 Dec 2008
17. John Thompson (2009) E-mail to Simon Lavington dated 10 Jan 2009