A NEW ASSEMBLY LANGUAGE FOR KDF9.

The new language, KAL4, bears many similarities to Usercode; the most apparent difference is that identifiers may be used to label both instructions and constants. An ALGOL-like block structure means that subroutines written as blocks can be transferred from program to program without causing label clashes. It is particularly convenient for programs handling ALGOL basic symbols. Programs are read from the disc, and translation is available via EDDON 2 and from paper tape.

These notes both describe the language (Sections A - E) and give details of the assembler. Corrections and additions, and details of library routines, will be in the file EDNKDOC-----.

The sections are as follows:

A: general rules and miscellaneous matters
B: instructions, labels, address values
C: block structure, scope, PROG
D: instructions
E: constants
F: assembler facilities
G: assembler output

For easy reference, the commoner numbered failures are given below; more details are given in Section G.

0 - ordinary error found
1 - end message found
2 - dictionary meets program
3 - stopped after 32 errors

Oct 1970 J.T.T.

A. GENERAL RULES.

Programs are read from PROMPT files on the disc. Space and tab are always ignored; newline is ignored only between complete items, and may not appear elsewhere. The line number on reference tables is such that line n is located by Ant*. All letters read in are treated as case normal, so that instructions may be written in either case, and references to an identifier may be made in either case. The symbol := is treated as : followed separately by =.

Identifiers must start with a letter. Any subsequent characters must be letters, digits, full stop, or ь. The first eight only are significant to the assembler.

As in Usercode, * denotes that assembly is to continue in a new word. Since constants may be assembled anywhere in a KAL4 program, * will usually be needed before a constant or its label. KAL4 has the additional facility that ** causes assembly to continue in a new 32-word lockout area. More than two asterisks will give the same effect as two only. The null instruction ; is ignored.

Comments starting with / or comment continue to the end of the line, and may contain any symbols. Comments starting with an opening round bracket continue until the next unmatched closing round bracket is reached. A terminating semicolon is not needed.

Library calls take the form library K1, K2, K101; . The case of the library letter is ignored, and the libraries, which are inserted in the order specified, need not all be different. The number of libraries in one call is limited to 41.

Translation stops on meeting an item consisting of an unmatched end . Any further contents of the program file are ignored.

B. INSTRUCTIONS AND LABELS.

Instructions are of 5 syntactic classes:

<function code>;	e.g. DUP;
<code><number>;	e.g. SHAD-32;
<code><number>/<number>;	e.g. FMMHNO/0;
<code>.<address>;	e.g. SET.fred;
<code><number>.<address>;	e.g. JCNZ15.loop;

<number> may be decimal or octal; decimal numbers are unsigned or have sign + or -; octal numbers must have the single sign ≠ ; e.g. SHL1; SHAD+1; SHC-16; PIBQ≠17; .

<address> is a list of address items, separated by + or -. The sign on the first or only item is optional. An address item is an identifier or an absolute address, the latter being an unsigned decimal number or ≠<octal number>, optionally followed by S<number> to give a syllable part,
e.g. FRED+1 JOE-037 A+B+≠20+C+D-≠1000S-1 .

The delimiters . / ; must always appear where shown in the instructions.

A point in the program may be labelled by an identifier followed by a colon. This is an implicit label declaration, the address value associated with the identifier being the current program address. An explicit label declaration can be used to associate any (16-bit) value with an identifier. It takes the form <identifier> = <address> : . Note the = (not :=) and : (not ;). e.g. after the declaration value=7: , SET.value, would set 7 in N1 and F.value; would bring the contents of word 7 to N1. Any identifier used on the right of an explicit label must be previously declared.

C. BLOCK STRUCTURE AND PROG.

The scope of an identifier is controlled by the brackets begin and end. A label declaration attaches a value to the identifier. It keeps this value between the previous unmatched begin and the corresponding end, unless it is re-declared in an inner block. It will not normally be available outside the block in which it was declared, so that subroutines written as blocks can use any labels for internal purposes without clashing with the main program.

For multi-entry subroutines, the label facility may be used. An identifier appearing in a label list is made accessible in the block immediately containing the block in which the label list occurs. The label list may occur anywhere; it consists of the symbol label followed by identifiers separated by commas, and is terminated by a semicolon. A failure will be given if the identifier actually is declared in the containing block.

e.g. label entry; label initiate, use, finish;.

A strict application of the block structure in the case of the right hand side of an explicit label would mean that, since forward references are not feasible, any identifier appearing there would have to have been declared earlier in the same block. To avoid this restriction, the block structure is ignored, and each identifier is given the value it would have if sufficient ends were inserted to make it accessible. This treatment also applies to identifiers used in segment items (see Section F).

If the identifier PROG is used on the left of an explicit label, the program address is moved to the address given by the right-hand side address part. If PROG appears in an address part, its value is the current address. This is always the first syllable of the present instruction or constant, so that e.g. J.PROG-OS1; skips back to before the preceding one-syllable instruction, but J.PROG+OS4; is used to skip the next one-syllable instruction, the extra syllables being those of the jump instruction itself. The implicit label PROG: is illegal.

e.g. PROG=PROG+GAP: PROG=PROG-OS1: =Q 0/PROG+1/PROG+32;

D.INSTRUCTIONS

The available instructions are detailed below. The initial contents of the nest are assumed to be

N1	N2	N3
a	b	c

and the state of the nest after execution of the instructions is given where appropriate.

Capital letters in the instructions are literals i.e. they stand for themselves.

n stands for any <number>

a stands for any <address>.

The kth and qth Q stores are referred to as Qk and Qq and the 16 bit counter, increment and modifier parts of the qth Q store are referred to as Cq, Iq and Mq respectively.

SET INSTRUCTIONS (3 syllables)

The allowed forms which result in the required value being placed in N1 are

SETn

SET.a

Note that n will be reduced to the range -32768 to +32767 and the sign bit will be extended from D32 to D0. SET.a places an address in N1 not the contents of the address.

EXIT INSTRUCTIONS (3 syllables)

Allowed forms are

EXIT.a	add a to LINK, jump to resulting address and delete L
EXITH.a	as above but add a further 3 syllables to address
EXITD	exit from director

LINK is the top cell of the SJNS

Note that a must be a whole word address.

EXIT and EXITH are equivalent to EXIT.0 and EXITH.0 resp. The usual subroutine return is EXITH.

DIRECTOR MODE INSTRUCTIONS (3 syllables)

FNSC	fetch nest number and stack count (K7)
FPHU	fetch PHU register (K5)
FRFIS	fetch RFI register and clock (K4)
SBANOL	store BA / NOL and CPL (=K1)
SBUZZ	switch buzzer on if N1>0 , off if N1=0 (=K0)
SCPDAR	store CPDAR (=K2)
SNSC	store nest number and stack counts (=K3)

Usercode equivalent is given in brackets.

MAIN STORE ADDRESSING

Main store instructions consist of F for fetch or S for store followed by a main store address which may be computed in a variety of ways.

DIRECT ADDRESSING (3 syllables)

Code	N1	N2	N3	Notes
F.a	x	a	b	x is the contents of address a
S.a	b	c		contents of N1 stored at address a
FMq.a	x	a	b	add a and Mq , fetch word from resulting address. $a \geq 0$
SMq.a	b	c		
FMQq.a	x	a	b	address computed as above. Iq added to Mq, 1 subtracted from Cq
SMQq.a	b	c		

INDIRECT ADDRESSING (2 syllables)

Code	N1	N2	N3	Notes
FMMq/k	x	a	b	add Mk and Mq , fetch word from resulting address.
SMMq/k	a	b		
FMMQq/k	x	a	b	address computed as above. Iq added to Mq, 1 subtracted from Cq
SMMQq/k	b	c		
FMMNq/k	x	a	b	the N causes 1 to be added to address computed as above.
SMMNq/k	b	c		
FMMQNq/k	x	a	b	
SMMQNq/k	b	c		
FMMHq/k	y	a	b	half word fetch. See note below.
SMMHq/k	b	c		
FMMQHq/k	y	a	b	
SMMQHq/k	b	c		
FMMHNq/k	y	a	b	compute address for half length fetch from Mq and Mk then add one word to it.
SMMHNq/k	b			
FMMQHNq/k	y	a	b	
SMMQHNq/k	b	c		

The address for half word fetching and storing is computed by taking half the contents of Mq and adding this to Mk. The most significant 24 bits of the word are taken if Mq is even and the least significant 24 bits if it is odd. The half word is transferred into or taken from the most significant end of N1 - hence y above has its least significant end zero.

When Qq forms part of the code the effect is to subtract 1 from the counter of the Q store and add the increment to the modifier after computing the fetch or store address.

ONE-SYLLABLE INSTRUCTIONS

CODE	FINAL STATE OF N1	N2	N3	NOTES
+	x	a	d	x=b+a
+F	x	a	d	
+D	x	y	a	x,y=ad+cb
+DF	x	y	a	
—	x	a	d	x=b−a
—F	x	a	d	
—D	x	y	a	x,y=cd−ab
—DF	x	y	a	
NEG	x	b	c	x=−a
NEGF	x	b	c	
NEGD	x	y	a	x,y=−ab
NEGDF	x	y	c	
×	x	a	d	x=a×b
×F	x	a	d	
×D	x	y	c	x,y=a×b
×DF	x	y	a	
×+F	x	y	a	x,y=a×b+cd
÷	x	a	d	x=b/a (−1 ≤x<1)
÷F	x	a	d	x=b/a
÷D	x	d	a	x=b,a/a (−1 ≤x<1)
÷DF	x	d	a	x=b,c/a
÷R	x	r	d	b,a=a×x+2^−47r (−1 ≤x<1)
÷I	r	x	c	b=a×x+r
ROUND	x	a	d	x=a,b ROUNDED TO SINGLE LENGTH
ROUNDF	x	c	d	
ROUNDH	x	b	c	x=a ROUNDED TO HALF LENGTH
ROUNDHF	x	b	c	
ABS	x	b	c	x=|a|
ABSF	x	b	c	
STAND	x	b	c	x=a IN STANDARD FLOATING FORM
FLOAT	x	c	d	x=b×2^a −128≤a≤+127
FLOATD	x	y	d	x,y=b,c×2^a −128≤a≤+127
FIX	x	y	b	y×2^x=a
MAX	x	y	c	x IS THE MAX. OF a & b, y IS THE MIN.
MAXF	x	y	a	OVERFLOW SET IF REVERSED
STR	x	y	b	y=a WITH DD=0, x=48 COPIES OF DD OF a.
CONT	x	a	d	x=b WITH THE SIGN DIGIT OF a
SIGN	x	a	d	x=+1 IF b>a x=0 IF b=a x=−1 IF b<a
SIGNF	x	a	d	
NOT	x	b	c	x=1's COMPLEMENT OF a
OR	x	c	d	x=a or b
AND	x	c	d	x=a and b
NEV	x	c	d	x=a nev b
BITS	x	b	a	x=NO. OF NON ZERO BITS IN a
ERASE	b	c	d	REMOVE a
ZERO	0	a	b	SET A ZERO WORD IN N1
DUP	a	a	b	DUPLICATE a
DUPD	a	b	a	DUPLICATE a,b. b in N4
REV	b	a	c	REVERSE a & b
REVD	c	d	a	REVERSE a,b & c,d. b in N4
PERM	b	a	a	INTERCHANGE abc CYCLICALLY
CAB	c	a	b	EQUIVALENT TO TWO PERMS
DUMMY	a	b	c	NO EFFECT
VR	a	b	c	CLEAR OVERFLOW REGISTER
SETTR	b	a	d	SET TEST REGISTER=DD OF a
TOB	x	c	d	x=a CONVERTED TO BINARY USING RADIX WORD b *
FRB	x	c	d	x=a CONVERTED FROM BINARY USING RADIX WORD b

*NOTE EACH DIGIT OF a MUST BE IN 6-BIT BINARY (NOT EXCESS 16 AS IN THE CHARACTER CODE).

MULT may be used instead of ×

DIV may be used instead of ÷

TWO-SYLLABLE INSTRUCTIONS

CODE	FINAL STATE OF N1	N2	N3	NOTES
SHA±n	x	b	c	$a=a\times 2^{\pm n}$
SHACq	x	b	c	$x=a\times 2^{Cq}$
SHAD±n	x	y	c	$x,y=a,b\times 2^{\pm n}$
SHADCq	x	y	c	$x,y=a,b\times 2^{Cq}$
				OVERFLOW SET IF SIGN CHANGES DURING SHIFT
SHL±n	x	b	c	x=a OR x,y=a,b SHIFTED AS A BIT PATTERN WITH NO NUMERICAL ASSOCIATIONS
SHLCq	x	b	c	
SHLD±n	x	y	c	
SHLDCq	x	y	c	
SHC±n	x	b	c	x=a SHIFTED CYCLICALLY
SHCCq	x	b	c	
SLINK	b	c	d	TRANSFER D32-47 TO THE TOP CELL OF THE SJNS
FLINK	x	a	b	x=THE CONTENTS OF THE TOP CELL OF THE SJNS
x+	x	y	e	$x,y=(a\times b)+c,d$
x+±n	x	y	e	$x,y=(a\times b)\,2^{\pm n}+c,d$
x+Cq	x	y	e	$x,y=(a\times b)\,2^{Cq}+c,d$

NOTE: $-64\leq n\leq -63$ $-128\leq Cq\leq +127$
FOR CYCLIC SHIFTS $-48\leq n,Cq\leq +48$.

Q-STORE INSTRUCTIONS (TWO-SYLLABLE)

CODE	FINAL STATE OF N1	N2	N3	Q-STORE	NOTES
FQq	Qq	a	b	NO CHANGE	
SQq	b	c	d	a	
S+Qq	b	c	d	Qq+a	*
FCq	Cq	a	b	NO CHANGE	**
SCq	b	c	d	L(a)/Iq/Mq	
SRCq	b	c	d	L(a)/1/0	
S+Cq	b	c	d	x/Iq/Mq	x=Cq+L(a) *
nCq	a	b	c	x/Iq/Mq	x=—Cq
DCq	a	b	c	x/Iq/Mq	x=Cq—1
FIq	Iq	a	b	NO CHANGE	**
SIq	b	c	d	Cq/L(a)/Mq	
SRIq	b	c	d	0/L(a)/0	
S+Iq	b	c	d	Cq/x/Mq	x=Iq+L(a) *
IONEQq,IMONEQq	a	b	a	Cq/±1/Mq	
ITWOQq,IMTWOQq	a	b	a	Cq/±2/Mq	
FMq	Mq	a	b	NO CHANGE	**
SMq	b	c	d	Cq/Iq/L(a)	
SRMq	b	c	d	0/1/L(a)	
S+Mq	b	c	d	Cq/Iq/x	x=Mq+L(a) *
M±Iq	a	b	a	Cq/Ip/x	x=Mq±Iq
QFRQk/q	a	b	c	Ck	
CFRQk/q	a	b	c	Ck/Iq/Mq	
IFRQk/q	a	b	c	Cq/Ik/Mq	
MFRQk/q	a	b	c	Cq/Iq/Mk	
IMFRQk/q	a	b	c	Cq/Ik/Mk	
CMFRQk/q	a	b	c	Ck/Iq/Mk	
CIFRQk/q	a	b	c	Ck/Ik/Mq	

NOTE: L(a) = THE LEAST SIGNIFICANT 16 BITS OF A

$1 \leq a \leq 15$, $0 \leq \leq 15$

00=OO=IO=MO=0

* N1 IS NESTED DOWN TO N2 DURING THE EXECUTION OF THESE INSTRUCTIONS

** THE SIGN DIGIT, 032, IS EXTENDED UP TO 00

JUMP INSTRUCTIONS (THREE-SYLLABLE)

CODE	FINAL STATE OF N1	N2	N3	NOTES
J.a	a	b	c	JUMP TO ADDRESS a
JS.a	a	b	c	JUMP TO ADDRESS a LEAVING ADDRESS OF THIS INSTR. IN SJNS'
JV.a	a	b	c	JUMP IF OVERFLOW SET. CLEAR OVERFLOW
JNV.a	a	b	c	JUMP IF OVERFLOW NOT SET CLEAR OVERFLOW
JTR.a	a	b	c	JUMP IF TEST REGISTER SET. CLEAR TEST REGISTER
JNTR.a	a	b	c	JUMP IF TEST REGISTER NOT SET. CLEAR TEST REGISTER
JEN.a	a	b	c	JUMP IF NESTING STORE EMPTY
JNEN.a	a	b	c	JUMP IF NESTING STORE NOT EMPTY
JEJ.a	a	b	c	JUMP IF SJNS EMPTY
JNEJ.a	a	b	c	JUMP IF SJNS NOT EMPTY
JCZq.a	a	b	c	JUMP IF Cq is ZERO
JCNZq.a	a	b	c	JUMP IF Cq IS NOT ZERO*
J>Z.a	b	c	d	JUMP IF a>0
J≥Z.a	b	c	d	JUMP IF a≥0
J=Z.a	b	c	d	JUMP IF a=0
J≠Z.a	b	c	d	JUMP IF a≠0
J<Z.a	b	c	d	JUMP IF a<0
J≤Z.a	b	c	d	JUMP IF a≤0
J=.a	b	c	d	JUMP IF a=b
J≠.a	b	c	d	JUMP IF a≠b

JCNZSq is a special 2 syllable Jump for short loops.

With these instructions
> ≥ = ≠ < ≤ may be replaced by
GT GE EQ NE LT LE if desired.

INPUT INSTRUCTIONS (TWO-SYLLABLE

MONITOR TYPEWRITER	PAPER TAPE READER	PAPER TAPE PUNCH	CARD READER	MAGNETIC TAPE UNIT	LINE PRINTER	GENERAL FORM
READ TWQq	READ PRQq	L.I.V.	BINARY READ	FORWARD READ MFRQq	L.I.V.	PIAQq
READ TO E.M. TREQq	READ TO E.M. PREQq	L.I.V.	BINARY READ TO E.M.	FORWARD READ TO E.M. MFREQq	L.I.V.	PIBQq
CHARACTER READ	CHARACTER READ PRCQq	L.I.V.	BINARY CHARACTER READ	FORWARD READ	L.I.V.	PICQq
CHARACTER READ TO E.M.	CHARACTER READ TO E.M. PRCEQq	L.I.V.	BINARY CHARACTER READ TO E.M.	FORWARD READ TO E.M.	L.I.V.	PIDQq
READ	READ	L.I.V.	ALPHA NUMERIC READ	BACKWARD READ MBRQq	L.I.V.	PIEQq
READ TO E.M.	READ TO E.M.	L.I.V.	ALPHA-NUMERIC READ TO E.M.	BACKWARD READ TO E.M. MBREQq	L.I.V.	PIFQq
CHARACTER READ	CHARACTER READ	L.I.V.	ALPHA-NUMERIC CHARACTER READ	BACKWARD READ	L.I.V.	PIGQq
CHARACTER READ TO E.M.	CHARACTER READ TO E.M.	L.I.V.	ALPHA-NUMERIC CHARACTER READ TO E.M.	BACKWARD READ TO E.M.	L.I.V.	PIHQq

OUTPUT INSTRUCTIONS (TWO-SYLLABLE)

MONITOR TYPEWRITER	PAPER TAPE READER	PAPER TAPE PUNCH	CARD READER	MAGNETIC TAPE UNIT	LINE PRINTER	GENERAL FORM
WRITE TWQq	L.I.V.	WRITE PWQq	L.I.V.	WRITE MWQq	WRITE LPQq	POAQq
WRITE TO E.M. TWEQq	L.I.V.	WRITE TO E.M. PWEQq	L.I.V.	WRITE TO E.M. MWEQq	WRITE TO E.M. LPEQq	POBQq
CHARACTER WRITE	L.I.V.	CHARACTER WRITE PWCQq	L.I.V.	WRITE LAST BLOCK MLWQq	CHARACTER WRITE	POCQq
CHARACTER WRITE TO E.M.	L.I.V.	CHARACTER WRITE TO E.M. PWCEQq	L.I.V.	WRITE LAST BLOCK TO E.M. MLWEQq	CHARACTER WRITE TO E.M.	PODQq
L.I.V.	L.I.V.	CHARACTER GAP PGAPQq	L.I.V.	GAP MGAPQq		POEQq
L.I.V.	L.I.V.	WORD GAP	L.I.V.	WIPE MWIPEQq		POFQq

MANIPULATIVE INSTRUCTIONS (TWO-SYLLABLE)

MONITOR TYPEWRITER	PAPER TAPE READER	PAPER TAPE PUNCH	CARD READER	MAGNETIC TAPE UNIT	LINE PRINTER	GENERAL FORM
L.I.V.	L.I.V.	L.I.V.	L.I.V.	FORWARD SKIP MFSKOq	L.I.V.	PMAOq
NO EFFECT	SET TR IF 8 CHANNEL SET	NO EFFECT	SET TR IF RECHECK SWITCH OFF	SET TR IF TAPE AT BTC MBTOq	NO EFFECT	PMBOq
NO EFFECT	NO EFFECT	NO EFFECT	NO EFFECT	SET TR IF LAST BLOCK LEVEL PRESENT MLBOq	NO EFFECT	PMCOq
L.I.V.	L.I.V.	L.I.V.	L.I.V.	REWIND MRWDOq	L.I.V.	PMDOq
L.I.V.	L.I.V.	L.I.V.	L.I.V.	BACKWARD SKIP MBSKOq	L.I.V.	PMEOq
NO EFFECT	NO EFFECT	NO EFFECT	NO EFFECT	SET TR IF E.T.W. PRESENT METOq	NO EFFECT	PMFOq

OTHER PERIPHERAL OPERATIONS (TWO-SYLLABLE)

CODE	OPERATION
MANUALOq	SET DEVICE UNREADY
BUSYOq	SET TR IF DEVICE BUSY
PAROq	SET TR IF:— PARITY FAIL (P/T. READ OR M/T) CHECK FAIL (CARD READ) ERROR (PRINTER)
TLOOq	SET TR IF ANY PART OF THE MAIN STORE AREA IS LOCKED OUT
INTOq	INTERRUPT IF DEVICE BUSY

OUT ;THREE-SYLLABLE: ENTRIES TO DIRECTOR

N1	N2	N3	NOTES
ZERO OR EMPTY	NOT USED	NOT USED	ENDS PROGRAM. TERMINATES ALL PERIPHERAL TRANSFERS IMMEDIATELY. CLEARS NS & SJNS. CALLS FOR NEXT PROGRAM.
1	PROGRAM NUMBER		ENDS PROGRAM. DOES NOT DEALLOCATE PERIPHERALS OR CLEAR NESTING STORES. ENTERS THE PROGRAM GIVEN IN N1 AND N2.
2	TIME LIMIT	NOT USED	TERMINATES PROGRAM AS IN OUT 0; ENTERS THE PROGRAM ALREADY IN STORE AT 60, TAKING THE TIME LIMIT FROM N2.
3	NOT USED	NOT USED	PUTS THE TIME TAKEN SO FAR (SECONDS TO 23 I.P.) IN N1, AND RETURNS TO THE NEXT INSTRUCTION.
4	IDENTIFIER	NOT USED	ALLOCATES THE TAPE DECK CARRYING THE TAPE WITH THE GIVEN IDENTIFIER. THE DEVICE NUMBER IS LEFT IN N1.
5	INTEGER	NOT USED	ALLOCATES UNITS OTHER THAN MAGNETIC TAPE. THE INTEGER IN N2 DEFINES THE UNIT 1 8 CHANNEL PAPER TAPE PUNCH 2 PAPER TAPE READER 3 HIGH SPEED ON-LINE PRINTER 4 CARD READER 5 6 CHANNEL PAPER TAPE PUNCH THE DEVICE NUMBER IS LEFT IN N1.
6	DEVICE No.	NOT USED	DEALLOCATES THE DEVICE
7	DEVICE No.	NOT USED	DEALLOCATES A MAGNETIC TAPE WHICH IS LEFT LOADED.
8	q O/p/q	NOT USED	OUTPUT WORDS p+1 TO q INCLUSIVE IN THE STREAM WHOSE NUMBER IS N WORD p. (N.B. THE CONTENTS OF p ARE ALTERED IN EXECUTION).

E. CONSTANTS.

A KAL4 constant consists of equals sign, length code, type code, constant string, semicolon. The length code is S, T, H, D or empty. The length code is compared with the current address, which must be aligned on a half-word for H, or a whole word for D or empty. H and D indicate half or double-length constants; in string constants, the end of the string is padded with dummy characters until the correct half- or whole-word alignment is obtained. The [constant string] is according to the type, listed below: details are given under the relevant number. Note that the constant is assembled starting at the current program address, forming part of the object program, and so should normally be preceded by a constant or an unconditional jump instruction (e.g. EXITH;). The types of constant are:

1. A address type
2. F floating-point
3. P printer string
4. Q Q-store type
5. S 1-syllable
6. T 2-syllable
7. X or × 6-bit basic symbol string
8. [basic symbol string
9. empty type fixed-point number
10. ≠ octal number.

1. Address type. Length H or blank.
The A is followed by any address part, which is assembled in the last 16 bits of the constant, the rest being zero, e.g. =HA item;.

2. Floating-point. Length H, D, blank.
Any ALGOL number, assembled in standard floating form. E may be used for ₁₀. Single and half-length will be correctly rounded; double-length will be truncated, e.g. =DF-.01; =F₁₀9;. See 29 for precise details.

3. Printer string. Length H, blank, D.
Any sequence of characters is packed in printer code up to a semicolon. Characters are letters, digits, ₁₀ . + - / , : = := (2 chars) ( ) × , underline. e.g. =HP F(X);. Formal symbols are:
a - asterisk (pounds sign) c - newline d - dummy
e - end message n - case normal p - page throw s - space
t - tab u - underline x - case shift.

*q gives a semicolon at the end of the current word, padding with dummies if necessary. See 38 for details of formal symbols.

4. Q-store. No length code allowed.
The only permitted form is = Q <address>/<address>/<address>;
e.g. =Q endtable-table / 1 / table; .

5. S. (Combined length/type code).
Either <number> to be assembled in the next syllable, or a basic symbol string starting at the current address. e.g. =S 128;.

6. T. (Combined length/type code).
The only permitted form is =T<address>; e.g. =T value-3; .

7. X or x. Length H, blank, or D.
The string must be letters, digits, . ₨ + or -. The symbols are packed in 6 bits each, left justified, padded with zeros to the correct alignment. + and - are represented by octal 10, 11.

8. ⌊ . Length S, H, D, empty.
Any printable symbol other than ⌊ or * may appear in the string. All editing characters are ignored (except that they will terminate octal formal symbols) and the case of letters is preserved. The string is terminated by ⌊, which must be followed by the final semicolon. Formal symbols are:

a - asterisk	c - newline	d - dummy
e - end message	l - underline	q - string quote ⌊
s - space	t - tab	u - closing quote ⌊

z - underlined end message.
e.g. =⌊String⌊; . See 38 for details of formal symbols.

9. empty type. Length H, D, empty.
The number may be expressed as any ALGOL number; E may be used for ₨. If the number is terminated by /, this must be followed by an integer specifying the number of integral places required, this is the amount of positive shift required to make the units bit coincide with the sign bit. If integral places are not specified, the presumed values are: for length H, 23; D, 94; or 47, i.e. the number is stored as an integer. Half and single-length numbers will be rounded.
e.g. =0; =D17₨14; =-1/0; =.1/-2; . See 29 for further details.

10. ≠. Length H, empty.
An octal number is specified as an octal integer, optionally followed by / and a non-negative number n corresponding to the integral places in decimal constants; the constant is shifted until the units bit is Dn of the KDF9 word (presumed values 23 if half-length, or 47).
e.g. =H/07026402; =≠1/0; =≠0612010612010612; .

29. ALGOL number assembly.
The maximum number of significant digits is 29; an undetected error may occur if more than 70 digits appear in the combined integer and fraction parts of a number. Failures are reported on finding a syntax error, if arithmetic overflow occurs or the number would require a shift removing all the significant bits or a floating number could not be expressed in standard form, whether too big or too small.
Any loss of accuracy is caused during multiplication or division by powers of 10 to handle decimal places and exponents. 93-bit accuracy should be obtained with integers (including those with positive exponents) and the loss of accuracy should not be greater than 2 bits for each 12 decimal places in the number (when expressed without exponent). This amount of error will only show in double-length fixed-point constants, and possibly in some other extreme cases. The exponent or shift can be any <number> as defined for instructions.

38. Formal symbols in strings.

In both basic symbol and printer strings, formal characters are indicated by the symbol *. This must be followed by an optional unsigned integer and either a formal symbol or an octal character. An octal character is a normal octal number not exceeding /77 for printer and /377 for basic symbol strings. The unsigned number is used as a repetition factor, e.g. *10A would place asterisk characters in the next 10 character positions. The repetition factor must be between 1 and 32768. e.g. =[*3c This *s is *s a *s string. *c*e]; , =P IDENT *q; . In octal characters, the character value is the octal number, e.g. = P */12; is the radix word for binary/decimal conversion, = P */73; is an end file (no formal symbol provided) and 7 dummies, =S[*/304]; is the first syllable of a SET instruction, =[*300/0]; is 50 zero words. Formal symbol letters may be in either case. Note that in all three types of string, any number of characters may appear, and the string is assembled in just the space it requires.

F. ASSEMBLER FACILITIES.

Translation is obtained by the ELDON 2 PROGRAM and SEGMENT commands. For translation via paper tape, see the end of this Section. The mandatory options are KAL4 and an ST option. The store limit specified is used for the translation, the minimum requirement being 2400 words. This allows the user to assemble programs of virtually any size with correspondingly faster service for jobs needing only a small amount of store for translation.

Options.

O/PL sends translator output to the line printer, only the message KAL4 O.K. or KAL4 FAIL n coming to the Teletype. NO OUTPUT should only be obtained if an ST less than 2100 is used. If O/PL is not specified, all output, including reference tables if requested, is sent to the Teletype.

TL is taken as the time limit for the object program - only needed for load-and-go running, when the maximum allowed is 30 seconds. If no TL is specified, 30 seconds will be given.

TABLES is used to obtain reference table output of all implicit labels and the position of each begin and end.

TEST causes the assembled program to be entered by OUT 2. The ST specified for translation must be sufficient for running. If TEST is not specified, the binary program will be written to the disc.

B-block setting.

The B-block of the program is initially set as follows: initial jump to word 8; STO; TL as specified; identifier as input file, with last 3 characters changed to UPU or USU; even restart ZERO; OUT; odd restart MRWDQQ;, filler word -1. The date of translation is placed in word 6.

These default options are adequate for most programs, the inclusion of K0 or K3000 automatically giving a useful restart, but any parts may be changed using PROG= ... . The initial jump is at 030, even restart 480, odd restart 483, and the filler word is word 7. If the filler word is unchanged at the end of assembly, the binary output will be from word 8 up to the highest address reached during assembly; otherwise, the area defined by the I and M parts of word 7 will be output.

The segment facility.

This facility allows communication of labels between the various segments of a program, or gives a convenient way of expressing a START option. A segment item takes the form segment ident/lo/hi;. Ident is the 12-character identifier which overwrites words 2/3. Minus signs indicate that the character required is that already in the identifier. If fewer than 12 characters are given, the rest are assumed to be minuses, and ident may be null. Characters must be in the KAL4 identifier set, and there must not be more than 12. The address values lo and hi define the area required as segment.

The assembler checks that all updating in the area has been performed. The initial jump is changed to J.lo; and this jump, the modified identifier, and the segment filler word are left in the B-block. The writing of a segment does not now affect the object program. If segment is used in a program, the normal binary output of the whole program at the end is inhibited. If TEST is specified, or a translate error has occurred earlier in the program, the segment item is simply checked for validity, and a binary will not be output.

The initial value of PROG is 8. The assembler ensures that the first four characters of the identifier are correct for disc binaries; for load-and-go running they are not altered, so the third character will normally be T. The options TEXT and OPTIMISER should not be used, and O/PC need not be specified. O/P8 will have the desired effect, except when the assembler is entered from the foreground job queue, when the output will be to stream 30.

Assembly via paper tape.

The assembler is now available independently of ELDON 2. The translate message is read from paper tape, and the source program from the disc. At present it is necessary to precede the message by a call tape for the assembler, EDNKAL4-----.

The message has the form: →<MESSAGE><IDENT><JOBNO><OPTIONS>→

<MESSAGE> is PROGRAM or SEGMENT.

<IDENT> is the 12-character identifier of the source file. The identifier must be in case normal, and the first four characters must be given exactly; the rest may contain wild minuses or may be abbreviated by full stop.

<JOBNO> is + followed by one case normal character other than semicolon, followed by / and two digits.

<OPTIONS> is a list of the following:

; or WITH	treat future options as WITH options
,	ignored
OUT	treat future options as WITHOUT options
ST	take the following number as store limit
TL	take the following number as time limit
O/P8	send assembler output to str. 10
O/PL	send assembler output to str. 30
P/T BINARY	if translation is without TEST, punch any binary output on paper tape

OPTIMISER	
TRACE	
TEXT	
TABLES	output reference tables
KALA	no effect
TEST	enter assembled program by OUT 2
→	finish reading options and start assembly.

ST, TL, O/P8, O/PL and → are effective whether or not WITHOUT is in force. Editing characters are ignored, and input is skipped to the first ←, after which additional ← s will be ignored. <MESSAGE> and options may be in either case, and may be abbreviated using full stops. Monitor indication is given of invalid message or options. In the case of an option error, the assembler proceeds using the options successfully decoded so far. The ST option on paper tape applies to the object program; the store required by the assembler must be specified on the operating form.

G. ASSEMBLER OUTPUT.

The output is headed by the program identifier, date and time. Reference table entries are output immediately each label is found, so that error messages will appear in the appropriate position in the reference tables. Reference tables have the form

IDENT wwwww/s L.aaa

where wwwww/s is the word/syllable address of IDENT, in octal, and aaa is the line number of the declaration. Error messages have the form: ERROR AT wwwww/s L.aaa AFTER IDENT L.bbb . wwwww/s and aaa define the position where the error was found, and the last implicit label was IDENT on line bbb (if no implicit label has been passed, the identifier PROG is printed). This is followed by a display of the line containing the error, with three asterisks inserted immediately after the point where the error was detected. Certain errors give rise to more than one ERROR message, and update errors will be detected at the appropriate end. For update errors, the ERROR message is followed by IN UPDATING IDENT and the line number aaa is the line on which the invalid reference to IDENT appeared. Update errors are given for the following:

non-zero syllable in EXIT or main store instruction (usually * omitted before the label of a constant)

in SET instructions or constants, attempting to update when the partially-formed address contains a syllable part (or word > 8191) or where the identifier contains a syllable part and the address being updated is non-zero. In either case, the desired result is ambiguous, but can always be obtained by suitable re-coding.

Unlocated identifiers are only detected at the end of assembly, the form of error message being UNLOC IDENT L.aaa, where aaa is the line on which IDENT was referred to; this is given for each reference to the identifier. When updating is not complete for a segment, the ERROR message is preceded by segment NEEDS UPDATING .

After an error, other than an update error, the program address is rounded up to a whole word: this tends to reduce additional consequent failures. If the failure is catastrophic, the normal message has ERROR replaced by FAILS and the address and line number printed may not be significant.

Each time a binary is output to the disc, its identifier and size are printed, where the size is the minimum store needed to load the program, in decimal rounded up modulo 32.

When reference tables are requested, the line number of each begin and end will be given, and comments starting with comment rather than / will be copied to the output device. An extra new line is taken after the end of each block.

When translation finishes, the message KAIA O.K. or KAIA FAIL n is output. This message will be duplicated on the Teletype if O/PC is not in use, and is followed by the run time and the amount of storage used. The number printed as STORE NEEDED will normally be an appropriate ST option to use for translation, but is insufficient if FAIL 2 was obtained. The failure numbers are explained below:

FAIL 0	Translation finished normally, but an error was found.
FAIL 1	End message read. Either not enough ends or closing string quote omitted near the end.
FAIL 2	Dictionary meets program. Use a larger ST or improve the block structure.
FAIL 3	More than 32 errors found - no point in continuing.
FAIL 4,5	Even, odd restart. Machine/translator error or ST<2400.
FAIL 6	Addresses for a disc binary exceed store available. Probably programming error, e.g. setting up filler word with address containing syllable part (syllable parts are now ignored in segment items).
FAIL 7	Machine error in OUT26 or OUT29. Try again.
FAIL 8	Machine or system error in library index or library text. Probably requesting a library which is not available, though this should lead to ERROR message.
FAIL 9	Internal assembler failure. Tell somebody.
FAIL 10	Error in translate message (paper tape). See monitor.
FAIL 11	Text file not found (paper tape call).