ICL

Education
& Training

SFL

system function language

BASIC FUNCTIONS

ACCUMULATOR

LSS	Set ACS=32 and load (OV set 0)	62
LSD	Set ACS=64 and load (OV set 0)	64
LSQ	Set ACS=128 and load (OV set 0)	66
SLSS	Stack, set ACS=32 & load (OV set 0)	42
SLSD	Stack, set ACS=64 & load (OV set 0)	44
SLSQ	Stack, set ACS=128 & load (OV set 0)	46
L	Load (OV set 0)	60
SL	Stack and load (OV set 0)	40
ST	Store	48
LUH	Load upper half (OV set 0)	6A
STUH	Store upper half	4A
CYD	Copy DR (OV set 0)	12
RRTC	Read real time clock (OV set 0)	68

B

DEBJ	Decrement B and jump (OV set)	24
LB	Load (OV set 0)	7A
SLB	Stack and load (OV set 0)	52
STB	Store (OV set 0)	5A
ADB	Add to B (OV set)	20
SBB	Sub from B (OV set)	22
MYB	Multiply B (OV set)	2A
CPB	Compare (CC set)	26
CPIB	Compare and increment (CC, OV set)	2E
VMY	Dope vector multiply (OV set 0)	2C

DR

LD	Load (CC set)	78
SLD	Stack and load (CC set)	50
STD	Store	58
MODD	Modify	16
LDRL	Load relative (CC set)	70
LDA	Load address	72
LDTB	Load type and bound	74
LDB	Load bound	76
INCA	Increment address	14
VAL	Validate address (CC set)	10
SIG	Start Significance (CC set)	28

STACK CONTROL

LLN	Load LNB	7C
RALN	Raise LNB	6C
STLN	Store LNB	5C
LXN	Load XNB	7E
ASF	Adjust SF	6E
STSF	Store SF	5E

JUMPS

J	Jump	1A
JLK	J and Link	1C

BASIC FUNCTIONS cont. ...

JUMPS cont. ...

JCC	J on Cond. Code	02
JAT	J on Arith. Cond. True	04
JAF	J on Arith. Cond. False	06
CALL	Call Procedure	1E
EXIT	Exit from Proc.	38
ESEX	Escape Exit	3A

STORE TO STORE

MV	Move	B2
MVL	Move Literal	B0
INS	Conditional Insert	92
PK	Pack (OV set)	90
SUPK	Suppress and Unpack (CC set, OV set 0)	94
TCH	Table Check (CC set)	80
TTR	Table translate	A6
CHOV	Check overlap (CC set)	B4
ANDS	And Strings	82
ORS	Or Strings	84
NEQS	≠ Strings	86
CPS	Compare Strings	A4
SWEQ	Scan while equal	A0
SWNE	Scan while unequal	A2

EMULATION

COM	Compress	B6
EXP	Expand	96
COMA	Compress ACC (OV set 0)	98
EXPA	Expand ACC (OV set 0)	88

SEMAPHORE (Clear Slaves)

INCT	Increment and Test (CC set)	56
TDEC	Test and Decrement (CC set)	54

SUNDRIES

MPSR	Modify PSR	32
CPSR	Copy PSR	34
IDLE	Idle	4E
OUT	Interrupt	3C
ACT	Activate (privileged)	3E
DIAG	Diagnose (privileged)	18

INTEGER (FIXED POINT)

IAD	Integer add (OV set) 1927	E0
ISB	Int. subtract (OV set) "	E2
IRSB	Int. reverse sub. (OV set) "	E4
IMY	Int. multiply (OV set) 7537	EA
IMYD	Int. mult. double (OV set 0)	EC
IDV	Int. divide (OV set) 10350	AA
IRDV	Int. reverse div. (OV set) 10475	AC

BASIC FUNCTIONS cont. ...

INTEGER (FIXED POINT)

IMDV	Int. remainder div. (OV, CC set)	AE
ICP	Int. compare (CC set)	E6
ISH	Int. shift (OV, CC set)	E8
FLT	Float (OV set)	A8
CDEC	Convert to dec. (OV set 0)	EE

LOGICAL

UAD	Logical add (CC set, OV set 0)	C0
USB	Log. subtract (CC set, OV set 0)	C2
URSB	Log. reverse sub. (CC set, OV set 0)	C4
UCP	Log. compare (CC set)	C6
USH	Log. shift	CB
ROT	Rotate (OV set 0)	CA
SHS	Shift single length (OV set 0)	CC
SHZ	Shift while zero (OV set 0)	CE
AND	And (OV set 0)	8A
OR	Or (OV set 0)	8C
NEQ	≠ (OV set 0)	8E

REAL (FLOATING POINT)

RAD	Real add (OV set)	F0
RSB	Real subtract (OV set)	F2
RRSB	Real reverse sub. (OV set)	F4
RMY	Real multiply (OV set)	FA
RMYD	Real mult. double (OV set)	FC
RDV	Real divide (OV set)	BA
RRDV	Real reverse div. (OV set)	BC
RDVD	Real divide double (OV set)	BE
RCP	Real compare (CC set)	F6
RSC	Real scale (OV set)	F8
FIX	Fix (OV set 0)	B8

DECIMAL

DAD	Decimal add (OV set)	D0
DSB	Dec. Subtract (OV set)	D2
DRSB	Dec. rev. sub. (OV set)	D4
DMY	Dec. multiply (OV set)	DA
DMYD	Dec. mult. double (OV set 0)	DC
DDV	Dec. div. (OV set 0)	9A
DMDV	Dec. remainder div. (OV set 0, CC set)	9E
DRDV	Dec. reverse div. (OV set 0)	9C
DCP	Dec. compare (CC set)	D6
DSH	Dec. shift (OV set)	D8
CBIN	Convert to binary (OV set)	DE

PRIMARY FUNCTIONS

All functions are primary except those listed as secondary or tertiary below.

Primary functions are used:

- without a variant; and usually with a literal operand
- with a named operand and optionally one of the variants:
  - .M
  - .I
  - .MI
  - .D
- with a variant and, sometimes, a literal operand. Variants are shown in the table:

k' / k''	DIRECT	INDIRECT		
	0	1	2	3
0	mn N	mn.D N	mn.G N	mn.GB
1	UNASSIGNED			
2	mn.L N	mn.DL N	mn.IL N	mn.MIL N
3	mn.X N	mn.DX N	mn.IX N	mn.MIX N
4	mn.P N	mn.DP N	mn.IP N	mn.MIP N
5	mn.S N	mn.DS N	mn.IS N	mn.MIS N
6	mn.T	mn.DT	mn.IT	mn.MIT
7	mn.B	Unassigned	mn.D	mn.MD

SECONDARY FUNCTIONS

The 2900 series secondary functions are:

ANDS	EXP	NEQS	SWEQ
CHOV	INS	ORS	SWNE
COM	MV	PK	TCH
CPS	MVL	SUPK	TTR

They may be written in two ways:

without a variant and with zero or two operands

with a variant and with one or three operands.

If written *without a variant*, the number of bytes in the operation is the bound/length field of DR. The compiler uses the operands (if any) as the mask and literal/filler bytes, respectively. The example below shows both forms:

```
| MV
| MV      | X'D0', X'40'
```

A secondary function may be written *with a variant*. This must be .N (number), meaning that the first (or only) operand gives the number of bytes in the operation. The second and third (if any) are the mask and literal/filler bytes, respectively. The example below shows both forms:

```
| SWEQ.N  | 25
| MVL.N   | 50, 0, C' '
```

Note that the compiler subtracts one from the first operand before planting it in the instruction.

The compiler generates a 32 bit function if mask and literal/filler are supplied; otherwise, a 16 bit function. All operands must be literals.

TERTIARY FUNCTIONS

The 2900 series tertiary functions are listed below:

JAF
JAT
JCC

The tertiary functions may be written in two ways; these are:

without a variant and with two operands

with a variant and with one or two operands.

The former is nearly always used. The first operand gives a code label and the second a mask.

The mask operand (M) is a single *hexadecimal* digit with the following significance for JAT and JAF:

REAL	M=0,ACC=0	M=1,ACC<0	M=2,ACC>0
INTEGER	M=4,ACC=0	M=5,ACC<0	M=6,ACC>0
DECIMAL	M=8,ACC=0	M=9,ACC<0	M=A,ACC>0
B	M=C,B=0	M=D,B<0	M=E,B>0
Other	M=B,DRbound=0	M=F, OV=1	

And for JCC:

Mask	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Jumps if CC is	-	3	2	2 3	1	1 3	1 2	1 2 3	0	0 3	0 2	0 2 3	0 1	0 1 3	0 1 2	ANY

The SFL jumps which compile into tertiaries are:

JE	(JNE)	Jump if (not) equal (JCC)
JG		Jump if greater (JCC)
JGE		Jump if greater or equal (JCC)
JL		Jump if less (JCC)
JLE		Jump if less or equal (JCC)
JN	(JNN)	Jump if (not) negative, integer
JND	(JNND)	Jump if (not) negative, decimal
JNR	(JNNR)	Jump if (not) negative, real
JNB	(JNNB)	Jump if (not) negative B
JP	(JNP)	Jump if (not) positive, integer
JPD	(JNPD)	Jump if (not) positive, decimal
JPR	(JNPR)	Jump if (not) positive, real
JPB	(JNPB)	Jump if (not) positive B
JZ	(JNZ)	Jump if (not) zero, integer
JZD	(JNZD)	Jump if (not) zero, decimal
JZR	(JNZR)	Jump if (not) zero, real
JZB	(JNZB)	Jump if (not) zero B
JZDL	(JNZDL)	Jump if (not) zero DR length
JOV	(JNOV)	Jump if (not) overflow

TERTIARY FUNCTIONS cont. ...

You can set up a switch by using

name	SWITCH(n)	label 0, label 1, ... label (n−1)

and use it with a modified jump referencing the name:

LB	value
J.M	name

FUNCTION FORMATS

Functions are either 16 or 32 bits in length. They are split between function code and operand as follows:

There are three basic operand formats:

primary (used for computational and miscellaneous instructions)

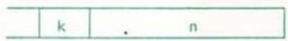
secondary (used for store-to-store instructions)

tertiary (used for jump instructions).

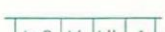
These basic formats break down further as follows:

PRIMARY FORMAT

16 bit



k	operand	
0	n	[7-bit signed literal]
1	.L n	[Direct access to local name space — n unsigned]
2	.IL n	[Indirect access via descriptor in local name space — n unsigned]
3	Implies further decode.	



k'' = 6 or 7, *reserved

32 bit

k'' = 0-5

Operands for values of k' and k'' given in table over.

FUNCTION FORMATS cont. ...

SECONDARY FORMAT

16 bit

32 bit

h = 0 Number of bytes = n+1
= 1 Number of bytes = Length of destination string
q = 0 16-bit instruction
= 1 32-bit instruction.

TERTIARY FORMAT

16 bit

k''' = 6 or 7

32 bit

k''' = 0-5

M = 4 bit mask field.

k'''	operand
0	N [literal]
1	.D N
2	.L N
3	.X N
4	.P N
5	.S N
6	.D } 16 bit
7	.MD } 16 bit

2900 BASIC FUNCTION CODES

SECOND HEXADECIMAL DIGIT \ FIRST HEXADECIMAL DIGIT	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		VAL	ADB	*	SL	SLD	L	LDRL	TCH	PK	SWEQ	MVL	UAD	DAD	IAD	RAD
2	JCC	CYD	SBB	MPSR	SLSS	SLB	LSS	LDA	ANDS	INS	SWNE	MV	USB	DSB	ISB	RSB
4	JAT	INCA	DEBJ	CPSR	SLSD	TDEC	LSD	LDTB	ORS	SUPK	CPS	CHOV	URSB	DRSB	IRSB	RRSB
6	JAF	MODD	CPB	*	SLSQ	INCT	LSQ	LDB	NEQS	EXP	TTR	COM	UCP	DCP	ICP	RCP
8	*	DIAG	SIG	EXIT	ST	STD	RRTC	LD	EXPA	COMA	FLT	FIX	USH	DSH	ISH	RSC
A	*	J	MYB	ESEX	STUH	STB	LUH	LB	AND	DDV	IDV	RDV	ROT	DMY	IMY	RMY
C	*	JLK	VMY	OUT	*	STLN	RALN	LLN	OR	DRDV	IRDV	RRDV	SHS	DMYD	IMYD	RMYD
E	*	CALL	CPIB	ACT	IDLE	STSF	ASF	LXN	NEQ	DMDV	IMDV	RDVD	SHZ	CBIN	CDEC	

EBCDIC CHARACTER SET

The following is the 95 character graphic set, with internal codes in hexadecimal:

Δ	40	d	84	H	C8
[	4A	e	85	I	C9
.	4B	f	86	}	D0
<	4C	g	87	J	D1
(	4D	h	88	K	D2
+	4E	i	89	L	D3
!	4F	j	91	M	D4
&	50	k	92	N	D5
]	5A	l	89	O	D6
$	5B	m	94	P	D7
*	5C	n	95	Q	D8
)	5D	o	96	R	D9
;	5E	p	97	\	E0
^	5F	q	98	S	E2
–	60	r	99	T	E3
/	61	~	A1	U	E4
¦	6A	s	A2	V	E5
,	6B	t	A3	W	E6
%	6C	u	A4	X	E7
_	6D	v	A5	Y	E8
>	6E	w	A6	Z	E9
?	6F	x	A7	0	F0
`	79	y	A8	1	F1
:	7A	z	A9	2	F2
£	7B	{	C0	3	F3
@	7C	A	C1	4	F4
'	7D	B	C2	5	F5
=	7E	C	C3	6	F6
"	7F	D	C4	7	F7
a	81	E	C5	8	F8
b	82	F	C6	9	F9
c	83	G	C7		

DIRECTIVES, CLASSIFIERS, DECLARATIVES

SFL Directives are:

MODULE
PROC
LPROC
SUBR
LSUBR
END

Classifiers are:

GDATA
DATA
} Can initialise items under these
CDATA
REDEF
PARAMS
LOCAL
CODE
LAYOUT

Data declaratives are:

INT	integer	32 bit
LINT	long integer	64 bit
WORD	word	32 bit
LWORD	long word	64 bit
LLWORD	long long word	128 bit
DEC	(packed) decimal	32 bit
LDEC	long decimal	64 bit
LLDEC	long long decimal	128 bit
REAL	real (floating point)	32 bit
LREAL	long real	64 bit
LLREAL	long long real	128 bit
BYTE	byte	8 bit
CHS	character string	8 bit
D	descriptor	64 bit

Listing control directives are:

TITLE
SUBTITLE
LIST
UNLIST

Alignment directives used to align items on a one, two or four word boundary, are

ALIGN or ALIGNS
ALIGND
ALIGNQ

You can abandon the compiler alignments using

NOALIGN

CTV's

CTV declared by

EQU Must assign value
AWORD
ABYTE(n)

CTV has form &name

AWORD, ABYTE can be altered by SET:

SET	&name, value

Compile time jumps given by:

AGO	'name
AIF	(cond), 'name
AIFNOT	(cond), 'name
CIF	(cond), 'name
CIFNOT	(cond), 'name

Note that AGO cannot jump backwards.

Expressions formed using operators:

+ – * / SL SR

= EQ LT GT LE GE NE

AND OR NEQ

Expressions must be in parentheses. They are evaluated L to R within types; type order is:

arith, then relational, then logical.

You can substring an ABYTE, using

name (firstchar, number)

You can concatenate CTV names with other names, character strings etc. But if the CTV is concatenated with a following letter, number, left parenthesis or solidus, they must be separated by a solidus, giving for instance &B/SB.

MACROS

Declared by

name	MACRO	
param0	name	param1, param 2, ...
	⋮	
	MEND	

To get a recursive macro you can use MACROR rather than MACRO in declaration. Parameter names have form &name, and are optional in the prototype line.

You refer to parameters in the macro body or prototype by position, as &0, &1, &2 etc, or by the names given in the prototype. You can give default values in the prototype line or in MDEFAULT:

param=val	mname	param=val, param=val, ...
	MDEFAULT	param, value

The default for a default is a null string.

When referring to a parameter by position (&3, say), you can replace the parameter number (3) by a further parameter or macro-time expression enclosed in parentheses, giving & (&N).

You can concatenate parameter names with other names, character strings, etc. But if the parameter is concatenated with a following letter, number, left parenthesis or solidus, they must be separated by a solidus, giving, for instance &A/AD.

You can cause immediate exit from a macro by giving:

	MEXIT	

CALL

Call a macro by giving a line of the same form as the prototype. If keywords are given in the prototype, you can use these (without &) in the call. If you don't use keywords, substitution is by position. If you mix keyword and positional values, positional ones substitute from the position given by the preceding keyword, if any.

You can put a variant or subscript after the macro name in the call, and use this within the macro by writing &'V.

MACROS cont. ...

SUBLISTS

You can give a bracketed list of values corresponding to a single parameter. You refer to these as param (0), param (1) and so on. You set default values for a sublist by giving param = (val, val...) in the prototype line, or

MDEFAULT	param, (val, val...)

If you refer to a sublist name without giving a number, you get the whole list, less brackets. To pass the sublist to another macro, you can enclose the sublist name in brackets to replace the brackets on the list passed.

MACRO VARIABLE FACILITIES

Corresponding to CTVs, you can use MEQU, MWORD, MBYTE, MSET, MGO. Note that MGO can jump backwards.

You can substring an MBYTE or parameter string using

&NAME (firstchar, number).

You can test MWORDS and MBYTEs using AIF, AIFNOT, CIF, CIFNOT. You can form expressions as for CTVs (and including CTVs, if you wish).

You can limit the number of jumps in a macro by writing

MCOUNT	(exp)

The default is 100.

MSCAN

You can scan an ABYTE, MYBTE or parameter within a macro to search for a given string and note its position. The format is:

MSCAN	&STRING,C'X',&POS,'NOTFOUND

&STRING is the string scanned. X is the string for which you are scanning. &POS is an AWORD or MWORD in which the position of the first character found is placed. 'NOTFOUND is a sequence symbol to which control is to be transferred if the string is not found; it is optional.

DOCUMENTATION

You can output error and warning messages:

MNOTE	0, C'WARNING MSG!'
MNOTE	1, C'ERROR MSG!'

You can insert comment for output on macro listing only, by starting a line with '*

MACROS cont. ...

MSWITCH

Can set up a switch using

&swname	MSWITCH(n)	'name0, 'name1, ...'name(n-1)

and jump (using MGO, AIF etc.) with
&swname (expression) replacing the jump destination.
You then jump to the 'name indicated by the expression.

ATTRIBUTE OPERATORS

Most can be used whenever a one-word integer literal is permitted. Operators are:

A'	Address type. See over.	
K'	Number of characters in ABYTE, MBYTE or macro parameter	
L'	Vector length	
P'	Relative address of descriptor	
R'	Relative address of data or code item	
T'	Type. See over.	
D'	Descriptor (2 word)	Instruction operands only
V'	Virtual address	
N'	Number of highest parameter supplied to macro, or number of items in sublist parameter. Macro use only.	

There are also system attribute operators:

&'G	Value of GDATA counter (bytes)
&'D	Value of DATA counter (bytes)
&'N	Value of named DATA counter (bytes)
&'X	Value of CDATA counter (bytes)
&'R	Value of REDEF counter (bytes)
&'L	Value of LOCAL counter (words)
&'C	Value of CODE counter (bytes)
&'P	Value of PLT counter (bytes) relative to final code descriptor for module
&'I	Value of IIL counter (bytes)
&'M	Invocation number of current macro
&'V	Variant type or Vector size in macro call
&'YX	Value of LAYOUT.X counter (bytes)
&'YL	Value of LAYOUT. L counter (words)
&'PLT	Returns 0 if in-line PLT, 1 if normal PLT.

VALUES FROM A'

Value	Symbol declared under
1	GDATA
2	DATA
3	Named DATA
4	CDATA
5	REDEF
6	PARAMS
7	LOCAL
8	CODE
11	LAYOUT.L
12	LAYOUT.X

VALUES FROM T'

Value (HEX)	Symbol is name of
X'0000'	Code (instruction)
X'0002'	External procedure
X'0100'	MODULE
X'0200'	PROC
X'0300'	LPROC
X'0400'	SUBR
X'0500'	LSUBR
X'0700'	Named DATA
X'0D01'	INT
X'0D02'	LINT
X'0D03'	WORD
X'0D04'	LWORD
X'0D05'	LLWORD
X'0D06'	DEC
X'0D07'	LDEC
X'0D08'	LLDEC
X'0D09'	REAL
X'0D0A'	LREAL
X'0D0B'	LLREAL
X'0D0C'	BYTE
X'0D0D'	CHS
X'0D0E'	D (descriptor)
X'0E00'	SWITCH
X'1001'	EQU
X'1002'	AWORD
X'1004'	ABYTE

DESCRIPTOR FORMATS

Type = 0 Vector descriptor

size – size of item

1 bit, size = 0, first byte = X'00' (if USC, BCI=0)
8 bit, size = 3, first byte = X'18' "
32 bit, size = 5, first byte = X'28' "
64 bit, size = 6, first byte = X'30' "
128 bit, size = 7, first byte = X'38' "

A = 0

USC – unscaled

= 0 When modifier added to address it is scaled according to size field, i.e. 2, 3 and 4 places up for 32, 64 & 128 bits. 3 places down for 1 bit.

BCI – Bound check inhibit

= 0 Any modifier added to address is checked (before scaling) to be less than Bound field.

Bound – If BCI = 0, bound should be 1 greater than the largest permitted modifier.

Type = 1 String descriptor

size, A,USC & BCI = 011000, first byte = X'58'

length – length in bytes of byte string whose first byte is addressed (possibly modified).

Type = 2 Descriptor descriptor

size, A = 1100, first byte = X'B0' (if USC, BCI = 0)

USC,BCI, Bound – as Type 0.

Type = 3 Miscellaneous

size, A USC, & BCI define subtype:

32 Bounded Code, first byte = X'E0'
33 Unbounded Code, first byte = X'E1'
35 System Call, first byte = X'E3'
37 Escape, first byte = X'E5'.

HEXADECIMAL AND BINARY CONVERSIONS

HEXADECIMAL TO DECIMAL

HEX	HEXADECIMAL COLUMNS					
	6	5	4	3	2	1
0	0	0	0	0	0	0
1	1,048,576	65,536	4,096	256	16	1
2	2,097,152	131,072	8,192	512	32	2
3	3,145,728	196,608	12,288	768	48	3
4	4,194,304	262,144	16,384	1,024	64	4
5	5,242,880	327,680	20,480	1,280	80	5
6	6,291,456	393,216	24,576	1,536	96	6
7	7,340,032	458,752	28,672	1,792	112	7
8	8,388,608	524,288	32,768	2,048	128	8
9	9,437,184	589,824	36,864	2,304	144	9
A	10,485,760	655,360	40,960	2,560	160	10
B	11,534,336	720,896	45,056	2,816	176	11
C	12,582,912	786,432	49,152	3,072	192	12
D	13,631,488	851,968	53,248	3,328	208	13
E	14,680,064	917,504	57,344	3,584	224	14
F	15,728,640	983,040	61,440	3,840	240	15

POWERS OF 2

2^n	n
256	8
512	9
1 024	10
2 048	11
4 096	12
8 192	13
16 384	14
32 768	15
65 536	16
131 072	17
262 144	18
524 288	19
1 048 576	20
2 097 152	21
4 194 304	22
8 388 608	23
16 777 216	24

POWERS OF 16

$2^0 = 16^0$	
$2^4 = 16^1$	
$2^8 = 16^2$	
$2^{12} = 16^3$	
$2^{16} = 16^4$	
$2^{20} = 16^5$	
$2^{24} = 16^6$	
$2^{28} = 16^7$	
$2^{32} = 16^8$	
$2^{36} = 16^9$	
$2^{40} = 16^{10}$	
$2^{44} = 16^{11}$	
$2^{48} = 16^{12}$	
$2^{52} = 16^{13}$	
$2^{56} = 16^{14}$	
$2^{60} = 16^{15}$	

16^n	n
1	0
16	1
256	2
4 096	3
65 536	4
1 048 576	5
16 777 216	6
268 435 456	7
4 294 967 296	8
68 719 476 736	9
1 099 511 627 776	10
17 592 186 044 416	11
281 474 976 710 656	12
4 503 599 627 370 496	13
72 057 594 037 927 936	14
1 152 921 504 606 846 976	15