Version 1, September 2009 **N2X3**

KDF9: Instruction set and instruction times.

Introduction to the KDF9's architecture.
The main memory consisted of 32768 48-bit words of core store.

There were 16 counter/index registers called *Q-stores*. Each was divided into three parts, Counter, Increment and Modifier. The counter could be tested for zero or non-zero by jump instructions. The modifier could be added to the address of main store references, in normal indexed addressing manner. All store reference instructions had a variant which incremented the Q-store after the main store reference, indicated by a letter `Q` at the end of the instruction mnemonic. In this case, after the main store access, the counter was decremented, and the modifier increased by the value held in the increment part. `Q0` was always zero.

The arithmetic and logic took place in the *nesting store*. At the normal user-program level there were 16 cells in a stack. A fetch from main store or a Q-store, pushed a 48-bit value onto this stack. A store instruction, popped the top value and put it in memory or Q-store. An immediate value could be put in the nesting store by a `SET` instruction. The `ZERO` instruction pushed a zero word onto this stack.

An attempt to put a 17th value in the nest led to a program failure called nest overflow, and generated an interrupt. Similarly an attempt to pop a value out of an empty nest generated an interrupt and underflow failure.

There were instructions to swap values around in the top 4 cells of the nest, and all arithmetic took place between operands in the upper cells of the nest. Thus the machine code was postfix.

The actual hardware of a time-sharing KDF9 actually had four such nesting stores, and four sets of Q-stores, but only the Director program saw any of this. The English Electric document of Director is on-line at: http://sw.ccs.bcs.org/KDF9/directorManuals/manuals.htm and gives a very good description of how this worked.

In addition there were 2 single-bit registers, the overflow register (V) and the test (TR) register. These were not quadruplicated, and had to be preserved by Director on interrupt, and restored on return to the program.

The `OUT` instruction was a programmed interrupt, permitting a program to call on the services of Director.

David Holdsworth.

KDF9 Machine Code Instructions.

N1 = top cell of nesting store
N2 = second cell, etc
N.B. All numbers in octal

Single syllable instructions

VR	001	clear overflow register
=TR	002	set test register
BITS	003	count number of bits in the word
×F	004	floating point multiply
×DF	005	floating point multiply – double length result from 48–bit operands
	006	
×+F	007	like ×DF but then followed by double length addition
NEGD	010	negate – double length
OR	011	inclusive or
PERM	012	permute top 3 nest cells, N1 becomes N3
TOB	013	convert six chars in N1 to binary number, radix word in N2
ROUNDH	014	round to half word
NEV	015	not equivalent, i.e. exclusive or
ROUND	016	round double number in N1,N2 to single in N1
DUMMY	017	do nothing as quickly as possible
ROUNDF	020	round floating point double number in N1,N2 to single in N1
ROUNDHF	021	round floating point number to half length
–DF	022	subtract double length floating point
+DF	023	add double length floating point
FLOAT	024	convert fixed point number to floating point
FLOATD	025	convert double length fixed point number to floating point
ABS	026	absolute value
NEG	027	negate
ABSF	030	absolute value floating point
NEGF	031	negate floating point
MAX	032	re–order N1, N2 to that larger is in N1
NOT	033	invert ones and zeroes
×D	034	multiply two 48–bit values to give double length result

×	035	multiply
–	036	subtract
SIGN	037	+1 if N1 – N2 > 0, –1 if N1 – N2 < 0, 0 if N1 – N2 = 0,
	040	
ZERO	041	put 0 in N1
DUP	042	duplicate, i.e. put copy of N1 in N1
DUPD	043	duplicate double length
÷I	044	integer divide, N1 = r*e*$_m$ainder, N2 = quotient
FIX	045	convert floating point to fixed point
	046	
STR	047	stretch 48–bit number to double length
CONT	050	convert double length integer in N1, N2 to single length in N1
REVD	051	swap N1 and N3, N2 and N4
ERASE	052	r*e*$_m$ove top cell of the nest
–D	053	subtract double length
AND	054	logical and
	055	
+	056	add
+D	057	add double length
÷	060	divide
÷D	061	divide double length
÷F	062	divide floating point
÷DF	063	divide double length floating point
÷R	064	fancy divide for multiplength division
REV	065	reverse, i.e. swap N1 and N2
CAB	066	permute top 3 nest cells, N3 becomes N1
FRB	067	convert binary to characters, radix in N2
STAND	070	standardise floating point number
NEGDF	071	negate double length floating point
MAXF	072	swap N1 N2 so that N1 is larger floating point
	073	
+F	074	add floating point
–F	075	subtract floating point
	076	
SIGNF	077	like SIGN but floating point

Q-stores

Q-stores are divided into three parts (like Cæsar's Gaul), counter, increment and modifier.
Memory addressing instructons containing M*q* add the value in the modifier to the address being accessed. For half word instructions half of one of the addresses is used.
When the instruction ends in Q, the counter is decremented, and the modifier incremented by the amount in the increment.

For peripheral instructions, the device number goes in the counter, the start address in the increment and the final address in the modifier.

Two syllable instructions

M*q*'M*q*	100	20*q* + *q*'	fetch 48–bit value in address *q* + *q*'
=M*q*'M*q*	101	20*q* + *q*'	store 48–bit value in address *q* + *q*'
M*q*'M*q*Q	102	20*q* + *q*'	fetch 48–bit value in address *q* + *q*' and increment Q*q*
=M*q*'M*q*Q	103	20*q* + *q*'	fetch 48–bit value in address *q* + *q*' and increment Q*q*
M*q*'M*q*H	104	20*q* + *q*'	fetch 24–bit value in address ½ *q* + *q*' to N1 top half and zeroise bottom half
=M*q*'M*q*H	105	20*q* + *q*'	store 24–bit value in address ½ *q* + *q*' N.B. top half of N1
M*q*'M*q*QH	106	20*q* + *q*'	fetch 24–bit value in address ½ *q* + *q*' and increment Q*q*
=M*q*'M*q*QH	107	20*q* + *q*'	store 24–bit value in address ½ *q* + *q*' and increment Q*q*
M*q*'M*q*N	110	20*q* + *q*'	fetch 48–bit value in address *q* + *q*' + 1
=M*q*'M*q*N	111	20*q* + *q*'	store 48–bit value in address *q* + *q*' + 1
M*q*'M*q*QN	112	20*q* + *q*'	fetch 48–bit value in address *q* + *q*' + 1 and increment Q*q*
=M*q*'M*q*QN	113	20*q* + *q*'	store 48–bit value in address *q* + *q*' + 1 and increment Q*q*
M*q*'M*q*HN	114	20*q* + *q*'	fetch 24–bit value in address ½ *q* + *q*' + 1
=M*q*'M*q*HN	115	20*q* + *q*'	store 24–bit value in address ½ *q* + *q*' + 1

M*q*'M*q*QHN	116	20*q* + *q*'	fetch 24–bit value in address ½ *q* + *q*' + 1 and increment Q*q*
=M*q*'M*q*QHN	117	20*q* + *q*'	store 24–bit value in address ½ *q* + *q*' + 1 and increment Q*q*
M+I*q*	140	20*q*	modifier of M*q* increased by value in I*q*
M–I*q*	141	20*q*	modifier of M*q* decreased by value in I*q*
NC*q*	142	20*q*	negate C*q*
DC*q*	143	20*q*	decrement C*q*
I*q*=+1	144	20*q*	I*q* = +1
I*q*=–1	145	20*q*	I*q* = –1
I*q*=+2	146	20*q*	I*q* = +2
I*q*=–2	147	20*q*	I*q* = –2
	150		
M*q*TOQ*q*'	151	20*q* + *q*'	copy M*q* to modifier of Q'
I*q*TOQ*q*'	152	20*q* + *q*'	copy I*q* to increment of Q'
IM*q*TOQ*q*'	153	20*q* + *q*'	copy I*q* and M*q* to increment and modifier of Q'
C*q*TOQ*q*'	154	20*q* + *q*'	copy C*q* to counter of Q'
CM*q*TOQ*q*'	155	20*q* + *q*'	copy C*q* and M*q* to counter and modifier of Q'
CI*q*TOQ*q*'	156	20*q* + *q*'	copy C*q* and I*q* to counter and increment of Q'
Q*q*TOQ*q*'	157	20*q* + *q*'	copy Q*q* to all of Q'
	160		
SHAC*q*	161	20*q*	shift arithmetic by number of bits in C*q*

SHA*n*	161	2*n* + 1	shift arithmetic by *n* bits
SHADC*q*	162	20*q*	shift arithmetic double length by number of bits in C*q*
SHAD*n*	162	2*n* + 1	shift arithmetic double length by *n* bits
×+C*q*	163	20*q*	×D; SHAC*q*; +D;
×+*n*	163	2*n* + 1	×D; SHA*n*; +D;
SHLC*q*	164	20*q*	shift logical by number of bits in C*q*
SHL*n*	164	2*n* + 1	shift logical by *n* bits
	165		
SHLDC*q*	166	20*q*	shift logical double length by number of bits in C*q*
SHLD*n*	166	2*n* + 1	shift logical double length by *n* bits
SHCC*q*	167	20*q*	shift cyclic by number of bits in C*q*
SHC*n*	167	2*n* + 1	shift cyclic by *n* bits
=M*q*	170	20*q* + 2	bottom 16 bits of N1 put in M*q*
=RM*q*	170	20*q* + 3	reset Q*q* to 0/1/0 then store N1 in M*q*
=I*q*	170	20*q* + 4	bottom 16 bits of N1 put in I*q*
=RI*q*	170	20*q* + 5	reset Q*q* to 0/1/0 then store N1 in I*q*
=C*q*	170	20*q* + 10	bottom 16 bits of N1 put in C*q*
=RC*q*	170	20*q* + 11	reset Q*q* to 0/1/0 then store N1 in C*q*
=Q*q*	170	20*q* + 16	all of N1 put in M*q*
M*q*	171	20*q* + 2	fetch M*q* into N1

I*q*	171	20*q* + 4	fetch I*q* into N1
C*q*	171	20*q* + 10	fetch C*q* into N1
Q*q*	171	20*q* + 16	fetch Q*q* into N1
=+M*q*	172	20*q* + 2	add value in N1 to M*q*
=+I*q*	172	20*q* + 4	add value in N1 to I*q*
=+C*q*	172	20*q* + 10	add value in N1 to C*q*
=+Q*q*	172	20*q* + 16	add value in N1 to Q*q*
LINK	173	0	fetch top call of SJNS into N1
=LINK	174	0	store N1 into top call of SJNS
JC*q*NZS	177	20*q*	jump to start of previous word if C*q* is non–zero
=Kn	175	2^{7-n}	set special register – director–mode only
Kn	176	2^{7-n}	fetch special register – director–mode only
=K0	175	200	if N1 = 0 then switch buzzer on else switch buzzer off
=K1	175	100	copy bits N1:D24–D33 to NOL, bits N1:D34–D35 to CPL and bits N1:D38–D47 to BA
=K2	175	40	copy bits N1:D32–D47 to CPDAR, where N1:D32 corresponds to buffer 15 and N1:D47 to buffer 0
=K3	175	20	switch to a new Q store/nest/SJNS set, with new nest depths (see below)
K4	176	10	push CLOCK/RFIR onto nest.
K5	176	04	push PHU onto nest
K7	176	01	push the current register set number and nest depths, as represented for the =K3 instruction.

The =K3 instruciton requires special care (see P29 of director listing).
N1:D0-D1 are the new register set number, N1:D2-D6 are the new nest depth and N1:D7-D11 are the

new SJNS depth; the register set number is independent of the program priority level.
The =K3 instruction must be followed by at least 6 DUMMY instructions, since it takes 6μsec to take effect and during this period the machine is in an indeterminate state.

The K5 instruction fetches bits D6-11 of the program hold up registers
PHU0:D6-D11 in N1:D0-D5,
PHU1:D6-D11 in N1:D6-D11,
PHU2:D6-D11 in N1:D12-D17,
PHU3:D6-D11 in N1:D18-D23.

Peripheral instructions (two syllable)

The unit number of the peripheral is in C*q* unless otherwise stated.
Start of store area address is in I*q* and end is in M*q*

CTQ*q*	120	20*q*	clear transfer – director–mode only
MANUALQ*q*	120	20*q* + 1	set peripheral unready
BUSYQ*q*	120	20*q* + 2	test if peripheral is busy
MLBQ*q*	120	20*q* + 4	set test register if previous read was a last block
MBTQ*q*	120	20*q* + 10	set test register if at begi*n*ing of tape
PARQ*q*	121	20*q*	test if peripheral has parity fail set
METQ*q*	122	20*q*	test if peripheral has end tape set (tape deck)
MFRQ*q*, PRQ*q*	124	20*q*	forward read
CLOQ*q*	124	20*q* + 2	clear lock–outs over area specified by I*q*–M*q* – director–mode only
TLOQ*q*	124	20*q* + 4	test for lock–out over area specified by I*q*–M*q*
PRCQ*q*	124	20*q* + 10	read paper tape, all 8 holes to each 48–bit word
PREQ*q*	125	20*q*	forward read to end message character
PRCEQ*q*	125	20*q* + 10	read paper tape to end message character, all 8 holes to each 48–bit word
MBRQ*q*	126	20*q*	backward read

MBREQ*q*	127	20*q*	backward read to end message character
PWQ*q*, MWQ*q*	130	20*q*	write
MLWQ*q*	130	20*q* + 10	write followed by tape mark, i.e. write a last block
MGAPQ*q*	130	20*q* + 14	leave a gap on mag tape
PGAPQ*q*	130	20*q* + 14	punch blank paper tape tape
MWIPEQ*q*	130	20*q* + 4	leave a really big clear gap on mag tape
MWEQ*q*, PWEQ*q*	131	20*q*	write to end message character
MLWEQ*q*	131	20*q* + 10	write to end message character followed by tape mark, i.e. write a last block
MFSKQ*q*	134	20*q*	forward skip one block
INTQ*q*	134	20*q* + 2	if thie device is busy suspend execution of this process until any peripheral transfer finishes
MBSKQ*q*	136	20*q*	backward skip one block
MRWDQ*q*, PRWDQ*q*	136	20*q* + 10	rewind
PIAQ*q*	124	20*q*	ordinary read
PIBQ*q*	125	20*q*	read to end–message
PICQ*q*	126	20*q*	which sort of read ? backward? definit*e*/y FH on disc
POAQ*q*	130	20*q*	ordinary write
POBQ*q*	131	20*q*	write to end–message
POCQ*q*	131	20*q*	don't know
PMAQ*q*	134	20*q*	seek on disc MFSK
PMBQ*q*	120	20*q* +	test MBT

		10	
PMCQ*q*	120	20*q* + 4	test MLB
PMEQ*q*	136	20*q* + 10	MRWD
PMFQ*q*	122	20*q*	MET
PMHQ*q*	124	20*q* + 6	SET lock outs
PMKQ*q*	134	20*q* + 4	don't know
PMLQ*q*	136	20*q* + 4	don't know

Jump instructions (three syllable)

The destination address is given in English Electric literature as *e*/*s*, where *e* is the address of the destination word, and *s* is the number of the syllable (0-5) within in the word. Instructions have to be in the bottom 8192 words of the address space, so *e* is at most 13 bits.

This address is spread throughout the 3 sylables of the instruction. *e* is broken down into
- e_l the least significant 8 bits of *e*
- e_m the next 4 significant bits of *e*
- e_h the single most significant bit

JE(*e*/*s*)=	$220 + 10e_h + s$	$20 + e_m$	e_l	jump if N1 = N2 and erase N1
JE(*e*/*s*)≠	$200 + 10e_h + s$	$20 + e_m$	e_l	jump if N1 ≠ N2 and erase N1
JE(*e*/*s*)<Z	$220 + 10e_h + s$	$40 + e_m$	e_l	jump if N1 < 0 and erase N1
JE(*e*/*s*)≥Z	$200 + 10e_h + s$	$40 + e_m$	e_l	jump if N1 ≥ 0 and erase N1
JE(*e*/*s*)>Z	$220 + 10e_h + s$	$100 + e_m$	e_l	jump if N1 > 0 and erase N1
JE(*e*/*s*)≤Z	$200 + 10e_h + s$	$100 + e_m$	e_l	jump if N1 ≤ 0 and erase N1
JE(*e*/*s*)=Z	$220 + 10e_h + s$	$140 + e_m$	e_l	jump if N1 = 0 and erase N1
JE(*e*/*s*)≠Z	$200 + 10e_h + s$	$140 + e_m$	e_l	jump if N1 ≠ 0 and erase N1

JE(*e*/*s*)V	220 + 10e_h + *s*	200 + e_m	e_l	jump if overflow is set
JE(*e*/*s*)NV	200 + 10e_h + *s*	240 + e_m	e_l	misprint in manual
JE(*e*/*s*)NV	200 + 10e_h + *s*	200 + e_m	e_l	jump if overflow is not set — probably true entry
JE(*e*/*s*)EN	220 + 10e_h + *s*	240 + e_m	e_l	jump if nesting store is empty
JE(*e*/*s*)NEN	200 + 10e_h + *s*	240 + e_m	e_l	if nesting store is not empty
JE(*e*/*s*)	200 + 10e_h + *s*	260 + e_m	e_l	jump unconditionally
JSE(*e*/*s*)	200 + 10e_h + *s*	320 + e_m	e_l	jump into a subroutine, address of nexxt instruction is pushed into the SJNS
JE(*e*/*s*)EJ	220 + 10e_h + *s*	300 + e_m	e_l	jump if SJNS is empty
JE(*e*/*s*)NEJ	200 + 10e_h + *s*	300 + e_m	e_l	jump if SJNS is not empty
JE(*e*/*s*)TR	220 + 10e_h + *s*	340 + e_m	e_l	jump if test register is set
JE(*e*/*s*)NTR	200 + 10e_h + *s*	340 + e_m	e_l	jump if test register is not set
JE(*e*/*s*)C*q*Z	240 + 10e_h + *s*	20*q* + e_m	e_l	jump if C*q* is zero
JE(*e*/*s*)C*q*NZ	260 + 10e_h + *s*	20*q* + e_m	e_l	jump if C*q* is non–zero
OUT	200	220	0	enter director – see below
EXITD	222	360	0	exit director — director–mode only
EXIT1 Ee	200	360 + e_m	e_l	exit subroutine
EXIT Ee	202	360 + e_m	e_l	exit subroutine
EXIT3 Ee	200	360 + e_m	e_l + 1	exit subroutine
EXIT2 Ee	202	360 + e_m	e_l + 1	exit subroutine

The EXIT instruction is very odd. EXIT1 means exit one half word after the address in the SJNS, and is the simple way out of a subroutine.

EXIT2 means exit two half words (i.e. one whole word) after the address in the SJNS, and is used as the normal exit from a subroutine, when that subroutine also has an error condition to indicate. In the case of error, the subroutine would do EXIT1 in the expectation that the next instruction was a jump to the error processing routine.

It was also occasionally used to implement a switch by placing a computed address in the SJNS, and using the EXIT *label* instruction where *label* was at the start of a list of jump instructions.

Data fetch and store instructions (three syllable)

The destination address is a 15-bit word address *e*.

This address is spread throughout the 3 sylables of the instruction. *e* is broken down into
- e_l the least significant 8 bits of *e*
- e_m the next 4 significant bits of *e*
- e_h the three most significant bits

For the SET instruction
- n_l the least significant 8 bits of *n*
- n_h the most significant 8 bits of *n*

E*e*	$300 + 10e_h$	e_m	e_l	fetch 48–bit word from absolute address
=E*e*	$301 + 10e_h$	e_m	e_l	store 48–bit word in absolute address
E*e*M*q*	$300 + 10e_h$	$20q + e_m$	e_l	fetch 48–bit word from *e* + M*q*
=E*e*M*q*	$301 + 10e_h$	$20q + e_m$	e_l	store 48–bit word in *e* + M*q*
E*e*M*q*Q	$302 + 10e_h$	$20q + e_m$	e_l	fetch 48–bit word from *e* + M*q* and increment Q*q*
=E*e*M*q*Q	$303 + 10e_h$	$20q + e_m$	e_l	store 48–bit word in *e* + M*q* and increment Q*q*
SET*n*	304	n_h	n_l	put the value *n* in N1

OUT instructions

The OUT instruction causes an entry into director, and its action thus depends on the director in use. However, there was always good consistency among the various incarnations of the time-sharing director. The following table is constructed from reading the code of the Eldon2 director from Leeds and the time-sharing director from Oxford.

The “out number” was in N1 and any other parameters in N2 etc. Obeying OUT with an empty nest was equivalent to OUT 0.

N1	N2	N3	Action
0	n/a	n/a	end the program

1	program name		load and enter a new code module whose name is in N2/N3
2	time limit	n/a	load and enter a new code module whose name is in N2/N3
3	n/a	n/a	put time used so far in N1
4	tape label	n/a	claim mag tape with 8 char label – unit number in N1
5	1 = paper tape punch 2 = paper tape reader 3 = line printer 4 = card reader	n/a	claim peripheral device – unit number in N1
6	unit number	n/a	deallocate peripheral device
7	mt unit number	n/a	deallocate mag tape, but leave loaded
8	params in Q–store format	n/a	transfer to the output well, C*q* is stream number
9	n/a	n/a	put clock–on–the–wall time in N1
10	tape label		claim mag tape with 16 char label – unit number in N1
14	?	?	someting to do with job accounting
15	?	?	someting to do with job accounting
17	n/a	n/a	put notional elapsed time in N2 and run time in N1
19	?	?	someting to do with job accounting
20	Q–/LO/HI (644 word buffer)	n/a	General text file location routine (NPL only) Enter with W0/1 of buffer = identifier Exit nest empty. Buffer containing W0= D0-23 Disc address of index block D24-47 Program address of entry - if found D24-47 0 - if not found W1= FOC library/FOC archive W3= mask W4-643= index block

25	Q–/LO/HI of prog now in store	n/a	restore dumped prog if D0=0, swop levels if D0=1 Used by JO to restart prog after JO has rolled it back into store
39	?	?	initiate foreground job
50	?	?	put stats block (job accounting) onto the OUT8 tape
51	?	?	put stats block (job accounting) onto the OUT8 tape

KDF9 Lineprinter Character Code

00	space
01	not used
02	line feed
03	page feed
04	tab ?
05	not used
06	%
07	'
10	:
11	=
12	(
13	)
14	£
15	*
16	,
17	/
20	0
21	1
22	2
23	3
24	4
25	5
26	6
27	7
30	8

31	9
32	not used
33	10
34	;
35	+
36	–
37	.
40	not used
41	A
42	B
43	C
44	D
45	E
46	F
47	G
50	H
51	I
52	J
53	K
54	L
55	M
56	N
57	O
60	P
61	Q
62	R
63	S
64	T
65	U
66	V
67	W
70	X
71	Y
72	Z
73	not used

74	not used
75	→
76	start message
77	ignored

KDF9 Paper Tape Character Code

Although the paper tape was 8-hole, it was used in a most curious way, and each row of holes produce only one 6-bit character in the machine.

```
○○○○○.○○○
s::p: :::
```

The holes marked with colons are the 6 bits that are transfered to the machine. The hole marked `p` is the parity bit. Parity is even. The hole marked `s` was only used in the space character so as to distinguish it from blank tape.

It was a two shift system, the cases being called shift and normal.

	normal	shift
00	space	
01		
02	CR–LF	
03		
04	tab	
05		
06	case shift	
07	case normal	
10		
11		
12		
13		
14		
15		
16		
17	/	:
20	0	^
21	1	[

22	2	]
23	3	<
24	4	>
25	5	=
26	6	×
27	7	÷
30	8	(
31	9	)
32	underline	underline
33	10	£
34	;	;
35	+	≠
36	–	*
37	.	,
40	not used	not used
41	A	a
42	B	b
43	C	c
44	D	d
45	E	e
46	F	f
47	G	g
50	H	h
51	I	i
52	J	j
53	K	k
54	L	l
55	M	m
56	N	n
57	O	o
60	P	p
61	Q	q
62	R	r
63	S	s
64	T	t

65	U	u
66	V	v
67	W	w
70	X	x
71	Y	y
72	Z	z
73	not used	not used
74	not used	not used
75	→	→
76		
77	ignored	ignored

Peripheral instructions (two syllable) — a complete list ??

Documentation on the peripheral instructions that came later in the life of KDF9 is difficult to find. This complete(?) list of PI, PO and PM instructions was compiled with the help of Bill Findlay. There is good reason to believe that there were no more such instructions acceptable to English Electric's Usercode Compiler.

PIAQ*q*	124	000	ordinary read	MFRQ*q*, PFRQ*q*
PIBQ*q*	125	000	read to end–message	MREQ*q*, PREQ*q*
PICQ*q*	124	010	PRCQ*q*	PRCQ*q*
PIDQ*q*	125	010	PRCEQ*q*	PRCEQ
PIEQ*q*	126	000	MBRQ*q*	MBRQ*q*
PIFQ*q*	127	000	MBREQ*q*	MBREQ
PIGQ*q*	126	010	alpha–numeric char read on CR	
PIHQ*q*	127	010	alpha–numeric char read to → on CR	
PMAQ*q*	134	000	seek on disc	MFSKQ*q*
PMBQ*q*	120	010	test MBT	MBTQ*q*

PMCQ*q*	120	004	test MLB	MLBQ*q*
PMDQ*q*	136	010	MRWD	MRWDQ*q*
PMEQ*q*	136	000	MBSK	MBSKQ*q*
PMFQ*q*	122	000	MET	METQ*q*
PMGQ*q*	134	010	Read C–store	Bill's best guess
PMHQ*q*	134	004	Set lockout	Bill's best guess
PMKQ*q*	135	000	IBM Even parity skip forward	Bill's best guess
PMLQ*q*	137	000	IBM Even parity skip back	Bill's best guess
POAQ*q*	130	000	ordinary write	MWQ*q*, PWQ*q*
POBQ*q*	131	000	write to end–message	MWEQ*q*, PWEQ*q*, TWEQ*q*
POCQ*q*	130	010	PWCQ*q* and MLWQ*q*	MLWQ*q*
PODQ*q*	131	010	PWCEQ*q* and MLWEQ*q*	MLWEQ*q*
POEQ*q*	130	014	PGAPQ*q* and MGAPQ*q*	MGAPQ*q*, PGAPQ*q*
POFQ*q*	130	004	MWIPEQ*q*	MWIPEQ*q*
POGQ*q*	132	000	CP A/N; FD Next sector	Bill's confident guess
POHQ*q*	133	000	CP A/N, →; FD Next sector, →	Bill's confident guess
POKQ*q*	133	010	CP A/N, →, Character mode; FD Next sector, →, fixed heads	Bill's confident guess
POLQ*q*	132	010	CP A/N, Character mode; FD Next sector, fixed heads	Bill's confident guess