As with C directives, there is no line reconstruction of the text.

U. Unset Parameters directive

Un, where n is a decimal integer, causes the preset parameter Pn to be unset. Further, Un-m unsets from Pn to Pm inclusive.

Z. End routine directive

Z indicates the end of a routine. Usually this is not necessary, since a new R directive implies the end of the preceding routine; any program material between Z and the next R will be assigned to routine 0.

Chapter 6

THE REMAINING ACCUMULATOR INSTRUCTIONS

In Chapter 3 we described the accumulator and some of the basic accumulator instructions.

All the accumulator instructions operate on floating-point numbers. They may be divided into groups as follows:-

(a) Standardised rounded operations

(b) Standardised unrounded operations

(c) Unstandardised operations

(d) Test instructions.

The only standardised rounded instruction not so far introduced is

360 Standardise a, round am and check for exponent overflow — $am' = a$ QRE

6.1 Standardised Unrounded Operations

In these instructions, L is cleared before the operation, and after the operation the result is standardised as a double-length number in A. An interrupt occurs if the exponent overflows.

300	Add am and s	$a' = am + s$ QE
301	Subtract s from am	$a' = am - s$ QE
302	Negate am and add s	$a' = -am + s$ QE

The instructions are thus similar to 320-322 except that rounding does not occur.

The following instructions are like 300 and 301 except that L and Ls are not cleared initially.

They provide a limited form of double-length working; limited because the answer is only correct if $ay \leq sy$ (i.e. the exponent of s must not be less than the accumulator exponent)

310	Add s to a (pseudo double-length)	$a' = a + s$ QE (if $ay \leq sy$)
311	Subtract s from a (pseudo double-length)	$a' = a - s$ QE (if $ay \leq sy$)

Before two numbers are added or subtracted in the accumulator, the one with the smaller exponent is shifted down into L and its exponent increased accordingly until the two exponents are the same.

In the 310 and 311 instructions, if $a_y \leq s_y$ then a_x is shifted down correctly. If $s_y < a_y$ then s_x will be shifted down into L, and the original contents of L will be spoiled. In this case the definitions of 310 and 311 will be

310 $am' = am + s$, l spoiled. QE

311 $am' = am - s$, l spoiled. QE

Extracodes are provided for correct double-length working in all cases and these are described later.

Two store locations are needed to hold a double-length number and it is conventional to store both numbers as standardised numbers with the less-significant half always positive and with an exponent which is at least 13 less than that of the more-significant number.

The contents of the accumulator are

$$am + a1.8^{-13}$$

as the floating point number a1 has an exponent a_y which is 13 more than its true value.

The 355 instruction is provided to position a1 correctly.

355	Copy the special sign bit of L (ls) into all bits of M, then standardise	$a' = a1.8^{-13}$	Q
		$a' = a - am$ when $ls = 0$	Q

Example:

To store the accumulator contents for double-length working in locations 100 and 101

356	0	0	100	store am
355	0	0	J4	position a1 *
356	0	0	101	store $a1.8^{-13}$ (Q)

* Note: All accumulator instructions make a reference to the store and obtain a store operand, even if the function does not use it. Any store address within the program is of course allowed, but as operands are read from the fixed-store very much faster than from the core-store it is conventional to specify the first address in the fixed-store, J4, in such instructions.

Example:

Locations 100 and 101 contain two numbers to be regarded as a double-length number. Add this number into the accumulator, using locations 98 and 99 as working space.

356	0	0	98	store am
355	0	0	J4	position a1
320	0	0	101	add less-significant halves, QRE, in am.

356	0	0	99	store partial answer
324	0	0	98	replace original am
300	0	0	100	add most-significant halves, result is standardised but not rounded in a
356	0	0	98	store most-significant part of this
355	0	0	J4	position the rest
300	0	0	99	add in less-significant sum
310	0	0	98	final answer in a

(This program is extracode 1500)

The following instructions complete the standardised unrounded operations

340	Standardise a, check for exponent overflow	$a' = a$	QE
342	Multiply am by s, leaving the double-length product standardised in a	$a' = am.s$	QE
343	As 342 but multiply negatively	$a' = -am.s$	QE
366	Clear L, complement am if negative, and standardise	$a' = \|am\|$	QE
367	Clear L, copy the modulus of s to Am, and standardise	$a' = \|s\|$	QE

6.2 Unstandardised Instructions

The unstandardised instructions can be divided into four groups

(a) Those concerned with storing and loading the accumulator.

(b) Multiplications.

(c) Divisions.

(d) Miscellaneous.

6.2.1 The unstandardised instruction which store and load a are described below:-

356	Copy am into S	$s' = am$
357	Copy a1 (that is 1, 1s and ay) into S	$s' = a1$
346	Transfer am to S and clear A (a' = floating-point zero)	$s' = am$, $a' = 0$
347	Transfer a1 to S and clear L and Ls	$s' = a1$, $1' = 0$
344	Copy the argument and sign from S into L and Ls, leaving Am, including the exponent, unchanged	$1' = sx$
345	Copy the argument from S into L and the sign bit from S into Ls and all bits of M. Leave the exponents unchanged.	$1' = sx$, $m' =$ sign of s, $ay' = ay$
314	Copy s into Am leaving L and Ls unchanged	$am' = s$, $1' = 1$
315	Copy s negatively into Am leaving L and Ls unchanged (AO will be set for $s = -1.0$)	$am' = -s$, $1' = 1$ AO

6.2.2 Unstandardised Multiplication Instructions.

372	Multiply am by s and leave the double-length product unstandardised in A. Clear the sign bit of L. Check for exponent overflow and accumulator overflow	$a' = am.s$ $ls' = 0$ EAO
373	As 372 but multiply negatively	$a' = -am.s$ $ls' = 0$ EAO
352	Multiply am by s, leaving the double-length product unstandardised in A, and set the sign bit of L equal to the sign of the product. Check for E and AO	$a' = am.s$ $ls' =$ sign of m EAO
353	As 352 but multiply negatively	$a' = -am.s$ $ls' =$ sign of m EAO

352 and 353 are identical to 372 and 373 except that they set Ls instead of clearing it. These instructions (352 and 353) are intended to form the single-length product of two unstandardised integers and leave the mantissa in L with the correct sign in Ls; they can therefore be redefined as

352	$1' = m.sx$	E
353	$1' = -m.sx$	E

Note, however, that the exponent ay' will be applicable to the double-length product in A, and that the accumulator overflow will not be set when the product overflows into M, but only when the double-length product overflows.

6.2.3 The Division with Remainder Instructions.

There are three division instructions which give the quotient in L and the remainder in M. However, these instructions only operate correctly for numbers which obey certain conditions.

There is a large range of division and remainder extracodes provided, which use these instructions and ensure the required conditions are fulfilled. For most purposes, it is easier to use these extracodes rather than the basic instructions. The only exception to this rule is the use of the 375 instruction for division of positive fixed-point integers, and this special case will therefore be described first:-

A fixed-point integer c can be represented in a 48-bit word by a fractional mantissa $sx = c \times 8^{-n}$, where n is normally 12 or 13, and an exponent sy, usually + 0 or + n. Provided that they are positive and that the divisor has $sx < \frac{1}{2}$, which even with $n = 12$ allows integers up to 30,000 million, the 375 instruction can be used to divide one such number into another. The dividend should first be placed in L, with M clear: this places the dividend in Ax with an additional scale factor of 8^{-13}, and ensures $ax < |sx|$ provided that $sx \neq 0$. The simplest case of the 375 instruction may now be defined as follows:-

375 Fixed-Point Integer Division

Divide a by the modulus of s, placing the quotient in L and the remainder in M. a and s must satisfy $0 \leq ax < |sx| < \frac{1}{2}$; the remainder will then lie in the range $0 \leq m' < |sx|$.	$1' = a/|s|$ E $m' =$ remainder $(0 \leq ax < |sx| < \frac{1}{2})$ $0 \leq m' < |sx|$ $ay' = ay - sy$

After obeying the 375 instruction the remainder m' will be scaled by the same factor as the dividend, and it should therefore be assigned the same exponent. The quotient 1' will be an integer scaled down by 8^{-13} and it must be shifted up one octal place if it is required to store it with a scale factor of 8^{-12}.

Example:

Given two fixed-point integers c and d in A5 and 1A5, each stored with one octal digit after the point; e.g. c is stored with mantissa 8^{-12} c in A 5. Form the quotient and remainder of c/d, in the same form and with exponent 12, and store them in locations 7 and 8.

345	0	0	A5	$a_x' = 8^{-8}$ (8^{-12} c) (i.e. $m' = 0$ and $l' = 8^{-12}$ c)
375	0	0	1A5	$l' = 8^{-13}$ c/d $m' = 8^{-12}$ x remainder
121	124	0	12U12	$a_y' = 12$
356	0	0	8	Remainder am to location 8
364	0	0	J4	$l' = 8^{-12}$ c/d
357	0	0	7	Quotient al to location 7

The full definition of the 375 instruction is as follows:

375 Pseudo Fixed-Point Division

Divide a by the modulus of s, placing the quotient in Al and a form of "remainder", m', in M. If $m' \geq 0$ the true remainder $m'' = m'$, but if $m' < 0$, which will only occur when $m'' \geq \frac{1}{2}$, then $m'' = m' + 1.0$	$al' = a/\|s\|$ E m' = "remainder" $(0 \leq a_x < \|s_x\| < 1)$ True remainder $m'' = m'$ if $m' \geq 0$ $= m' + 1.0$ if $m' < 0$ $0 \leq m'' < \|s_x\|$

a and s need not be standardised but they must satisfy the restriction $0 \leq a_x < |s_x| < 1.0$. The true remainder m" will then satisfy the constraint $0 \leq m'' < |s_x|$. a_y' is the exponent of the quotient. The exponent of the remainder is $a_y - 13$, i.e. 13 less than that of the double-length dividend a before the operation.

If $s_x \leq a_x < 8\,|s_x|$ or if $s_x = -1.0$ the 375 instruction provides a quotient and remainder m" which are correct if regarded as floating-point numbers but which break the rules of fixed-point division. The remainder may be larger or smaller than with true fixed-point division, its exponent being as follows:

Condition	Exponent of Remainder
$\|s_x\| \leq a_x < 8\,|s_x|$	$a_y - 12$
$0 \leq a_x < \|s_x\| < 1.0$	$a_y - 13$
$s_x = -1.0$ and $a_x \geq \frac{1}{8}$	$a_y - 13$
$s_x = -1.0$ and $a_x < \frac{1}{8}$	$a_y - 14$

When $|s_x| \leq a_x$ the adjusted remainder m" may not be exact, because the last octal digit of the correct remainder will have been lost.

If $a_x < 0$ then am will be negated before the division takes place but l will not be adjusted.

If $a \geq 1.0$ then a_x will be shifted down and its exponent increased by one before the operation.

If $s_x = 0$ or $a_x \geq 8\,|s_x|$, the 375 instruction will not give a correct quotient or remainder.

376 Divide a by the modulus of s, placing the quotient in Al and the "remainder" in M. The dividend a must not be negative and the divisor s must be standardised before the 376 instructed is obeyed. The "remainder" is such that:

$al' = a/|s|$ $(a \geq 0)$. EDO

Remainder
$m'' = m'$ if $m' \geq 0$
$= m' + 1.0$ if $m' < 0$

mantissa of true remainder $= m$ if $m \geq 0$
$= m + 1.0$ if $m < 0$

exponent of true remainder $= a_y - 13$ if $|a_x| < |s_x|$
$= a_y - 12$ if $|a_x| \geq |s_x|$

The quotient al' is not normally an integer; it is merely the unrounded representation of a/s to such accuracy as is possible in the 39 binary digits of L. The true remainder has no special significance other than that it represents a - s.al' and is always positive or zero. When $|a_x| \geq |s_x|$ the true remainder m" may not be exact, because the last octal digit of a - s.al' will have been lost.

Exponent overflow is checked for, and division overflow occurs if s is unstandardised or zero. If a is in standardised form before the division, al' will be a standardised quotient, but m' and m" may not be standardised.

377 Divide the modulus of am by the modulus of s, placing the quotient in Al and the "remainder", as defined for 376, in M. Check for E and DO. The divisor s must be standardised. If am is in standardised form before the division, al' will be a standardised quotient, but m' and m" may not be standardised.

$al' = |am| / |s|$ EDO

Remainder
$m'' = m'$ if $m' \geq 0$
$= m' + 1.0$ if $m' < 0$

6.2.4 Miscellaneous Unstandardised Instructions

354	Round by adding. Add one to the least-significant digit of m if the most-significant digit of l is a one. Accumulator overflow can occur. The contents of L are unchanged.	$am' = a.R+$ AO
341	Check for exponent overflow. a is unchanged.	$a' = a$ E
361	Round am and check for exponent overflow.	$am' = a$ RE

6.3 Test Instructions

The following four instructions are tests on the accumulator mantissa, and comparable to the tests on bt or bm. Note that bm can be used to modify the address.

234	Place n in Ba if the accumulator contains zero.	ba′ = n if $a_x = 0$
235	Place n in Ba if the accumulator does not contain zero.	ba′ = n if $a_x \neq 0$
236	Place n in Ba if the accumulator contents are greater than or equal to zero.	ba′ = n if $a_x \geq 0$
237	Place n in Ba if the accumulator contents are less than zero.	ba′ = n if $a_x < 0$

All these test ignore the sign bit of L.

For the accumulator to contain zero, both guard bits must be zero; the most-significant guard bit, rather than the sign bit, determines whether the accumulator is greater or less zero. With standardised numbers this is immaterial, as the guard digits will be copies if the sign bit, and with fixed point working the correct result might still be obtained even if accumulator overflow had occured.

Examples:

1. Increase b3 by 0, 1 or 2 depending on whether am is >, = or < the contents of store location 16. Let A10 be the address of a register available for working space.

356	0	0	A10	store am
321	0	0	16	am - s
234	3	3	1	b3′ = b3 + 1 if am = s
237	3	3	2	b3′ = b3 + 2 if am < s
334	0	0	A10	restore am

2. B1 and B2 contain positive integers n1 and n2. Form n1 × n2 in store location 5 as a fixed-point integer, represented by mantissa n1 × n2 × 8^{-12} and zero exponent. Replace b1 by the integer quotient n1/n2, and place the remainder from this division in B2. Let locations 6 and 7 be available for working space.

113	1	0	6.4	set b1, b2 in the store
113	2	0	7.4	as floating-point numbers
113	0	0	6	with zero exponents
113	0	0	7	
334	0	0	6	am′ = n1 × 8^{-12}
352	0	0	7	a′ = n1 × n2 × 8^{-24}
365	0	0	J4	a′ = n1 × n2 × 8^{-25} i.e. l′ = n1 × n2 × 8^{-12}
357	0	0	5	store l = n1 × n2 × 8^{-12}

345	0	0	6	set n1 in L with m′ = sign of n1 = 0. a_x′ = n1 × 8^{-25}
375	0	0	7	l′ = (n1/n2) × 8^{-13}, m′ = remainder × 8^{-12}
356	0	0	6	store remainder am
364	0	0	J4	shift up quotient to integer position. l′ = (n1/n2) × 8^{-12}
357	0	0	7	store quotient l = (n1/n2) × 8^{-12}
101	2	0	6.4	set b2′ = remainder
101	1	0	7.4	set b1′ = quotient

Note that in this example it is not necessary to set the exponent zero after division because a_y is made zero during the multiplication and both division operands have zero exponents.

Chapter 7

EXTRACODE INSTRUCTIONS

7.1 Introduction

The basic instructions consist in just those simple operations which the computer has been designed to execute directly. In the Atlas order-code, however, there are many complicated operations which the computer deals with in a special way; these are known as extracodes and are distinguished from the basic instructions by having a 1 in f_0, the most-significant bit of the 10-bit function number. Upon encountering an instruction with $f_0 = 1$, there occurs an automatic entry to one of many built-in subroutines, the choice being determined by the remaining three octal digits of the function number. The exit from the subroutine is again automatic, and the program proceeds in the usual way with the instruction next after the extracode, unless the extracode subroutine has initiated a jump.

7.1.1 Uses of the Extracode Instructions.

As their name implies, the extracodes provide an extension of the basic order-code, including both those complicated operations which are excluded from the basic instructions, and many of the facilities which on previous machines have been obtained by the use of library subroutines.

Amongst the arithmetic instructions provided by extracodes we may instance those in which the address, interpreted as a floating-point number, is used as an operand; double-length operations; and a full range of elementary functions such as logarithm, square-root, sine etc.

An important group of extracodes deals with the special requirements of input and output and also of magnetic tape transfers; the uses of these will be discussed at some length in Chapters 8 and 9.

The organisational extracodes comprise extensive facilities designed to assist the programmer in making efficient use of the operating system of Atlas. The various aspects of this are described in later Chapters (particularly Chapters 11 and 12).

7.1.2 To the programmer, extracode instructions appear as basic instructions. The two types of instruction can be freely intermixed, and after each instruction control passes sequentially to the next (except for jump instructions). It is therefore not strictly necessary to know how the computer deals with extracode instructions, although this is given for completeness in the next section.

There are 512 function numbers available for extracodes, 1000-1777. Of these, 1000-1477 are singly-modified instructions (B-type) and 1500-1777 are doubly-modified instructions (A-type). In some of the B-type instructions, bm is used as an operand so no modification takes place.

7.2 The Logical Interpretation of Extracode Instructions

When an extracode instruction is encountered the following action takes place:-

(a) The content of Main control, b127, is increased by one to the address of the next program instruction.

(b) The address is modified according to the type (i.e. N + bm for B-type, N + ba + bm for A-type) and the result stored in B119.

(c) The seven Ba digits are placed in bits 15-21 of B121, unless Ba is B122 in which case B121 is left unchanged; this enables B122 to be used to specify a B-register in extracode functions exactly as in basic functions.

(d) The function digits f1 - f9 are placed in extracode control, B126, as shown below.

Bit	0	1-9	10	11	12	13	14	15	16	17	18	19	20	21-23
Value	1	0 0 0 0 0 0 0 0 0	f1	f2	f3	0	0	f4	f5	f6	f7	f8	f9	000

(e) Control is switched from Main (B127) to extracode (B126).

The next instruction to be obeyed is now in the fixed store, under extracode control, at a location determined by the function digits. It is in one of 64 registers (given by f4-f9) in one of 8 tables at intervals of 256 words (given by f1-f3). The tables of 64 registers are called "jump tables". In general this instruction will be an unconditional jump into a routine which performs the required function. These routines are permanently stored in the fixed-store and written in normal basic instructions. Each routine terminates with an instruction in which f1 = f3 = 1 in the function number. This is obeyed as if f1 = 0 and then control is switched back to main control (e.g. 521 is equivalent to 121 followed by "extracode exit"). The next instruction to be obeyed is then the one whose address is in B127; if no jump has been initiated by the extracode this instruction will be the one immediately following the extracode instruction.

The routines that perform extracodes can use B-registers 91 to 99 inclusive and always use B119, B126, and B121 (unless Ba = 122).

Examples:

1. Extracode 1714 is defined as am' = 1/s
Replace the numbers in locations 100 to 105 by their reciprocals.

```
 121     1     0     5     set modifier/count
2)1714   0     1     100   am' = 1/s
 356     0     1     100   store
 203     127   1     A2    count
```

Each time the extracode instruction is encountered b127' = b127 + 1, b121' = 0, b119' = 100 + b1 + b0, b126' = J40034140 = +1804J4 and control is switched to B126. The instruction in the jump table is

```
121   126   0   A14
```


The instructions at A14 are

```
14)334   0     0     A96   set am = +1
   774   0     119   0     374 division 1/s, then reswitch to
                           main control.
96)+1
```

2. Extracode 1341 is defined as ba' = ba.2^n (arithmetic shift up)
Shift b16 up by 2 more than the integer in B17

```
1341   16   17   2
```

This instruction sets b121' = 16D1, b119' = 2 + b17, etc. (Note that b16 is not added to b119 because 1341 is a singly-modified (B-type) extracode).

3. Shift the contents of B20 to B47 inclusive up by 5 places.

```
 121    121   0     20D1   set B121 pointing at B20.
1)1341  122   0     5      shift. As Ba = B122, b121 is left
                           unchanged when the extracode is
                           entered.
 172    121   0     47D1   bt' = b121 - 47D1
 220    127   121   A1     If bt ≠ 0, b121' = b121 + 0.4
                           and b127' = A1
```

Example 3 illustrates the use that can be made of B121 and B122 in extracodes; this is the same as their use in basic instructions except that extracodes with Ba ≠ 122 will overwrite B121.

7.3 Allocation of Functions

The extracodes are divided into sections as shown below, though there are a few functions which do not fit into this pattern. References are given for those subjects described in this chapter.

Functions	Subjects	Reference
1000-1077	Magnetic tape routines, and Input and Output routines.	--
1100-1177	Organisational routines.	--
1200-1277	Test instructions and 6-bit character operations	7.6 & 7.5.2
1300-1377	B-register operations.	7.5.1
1400-1477	Complex arithmetic, vector arithmetic and miscellaneous B-type accumulator routines.	7.4.6 & 7.4.7
1500-1577	Double-length arithmetic and accumulator operations using the address as an operand.	7.4.4 & 7.4.5
1600-1677	Logical accumulator operations and half-word packing.	7.5.3 & 7.4.8
1700-1777	Arithmetic functions (log, exp, sq.rt., sin, cos, tan, etc.) and miscellaneous A-type accumulator operations.	7.4.1, 7.4.2 & 7.4.3

Not all of the 512 extracode functions have been allocated and, where convenient, constants and extracode programs have been packed into the vacant jump-table locations.

This means that the use of an unallocated extracode function may result in an 'unassigned function' interrupt or may cause some extracode to be entered incorrectly. The latter case would give the programmer wrong results.

In particular, the first location in the fixed store, J4, contains the floating-point number $\frac{1}{2}$. This causes an unassigned function interrupt if extracode 1000 is encountered, since J4 is the first register of the first jump-table. Note that floating-point zero is equivalent to the instruction

```
1000    0    .0    0.
```

There follows a description of many of the extracodes. Where possible, the actual number of basic instructions obeyed in each extracode routine is given in the right hand column.

Appendix E gives an ordered summary of all the extracodes, for easy references.

7.4 The Accumulator Extracodes

7.4.1 The Most Used Arithmetic Functions

The following routines each have two extracode numbers. The first operates on s, which is standardised on entry. The second operates on a, which is standardised, rounded and truncated to a single-length number on entry. For this number we use the notation aq. The results are always standardised rounded numbers in Am.

1700	Place the logarithm to base e of s in Am.	$am' = \log s$	
1701	Place the logarithm to base e of aq in Am.	$am' = \log aq$	
1702	Place the exponential of s in Am.	$am' = \exp s$	43
1703	Place the exponential of aq in Am.	$am' = \exp aq$	42
1710	Place the square root of s in Am.	$am' = +\sqrt{s}$	≤42
1711	Place the square root of aq in Am.	$am' = +\sqrt{aq}$	≤41
1712	Form the square root of $(aq^2 + s^2)$ and place this in Am.	$am' = +\sqrt{aq^2 + s^2}$	≤50

Following the two arc sine extracodes, am' is in radians, with $-\frac{\pi}{2} \leq am' \leq +\frac{\pi}{2}$

1720	Place the arc sine of s in Am.	$am' = \text{arc sin } s$
1721	Place the arc sine of aq in Am.	$am' = \text{arc sin } aq$

Following the two arc cosine ex tracodes, am' is in radians, with $0 \leq am' \leq \pi$

1722	Place the arc cosine of s in Am.	$am' = \text{arc cos } s$
1723	Place the arc cosine of aq in Am.	$am' = \text{arc cos } aq$

Following the two arc tangent extracodes, am' is in radians, with $-\frac{\pi}{2} < am' < \frac{\pi}{2}$

1724	Place the arc tangent of s in Am	$am' = \text{arc tan } s$
1725	Place the arc tangent of aq in Am.	$am' = \text{arc tan } aq$
1726	Divide aq by s and place the arc tangent of this number in Am. am' is in radians and such that $-\pi < am' \leq \pi$.	$am' = \text{arc tan } (aq/s)$
1730	*Place the sine of s in Am.	$am' = \sin s$ 41
1731	*Place the sine of aq in Am.	$am' = \sin aq$ 40
1732	*Place the cosine of s in Am.	$am' = \cos s$ 42
1733	*Place the cosine of aq in Am.	$am' = \cos aq$ 41
1734	*Place the tangent of s in Am.	$am' = \tan s$ 34
1735	*Place the tangent of aq in Am.	$am' = \tan aq$ 33

* In 1730 -1735, s and aq must be in radians.

7.4.2 Other Floating-Point Arithmetic Functions

1704	Place the integer part of s in A.	$a' =$ int pt s	QE	5
1705	Place the integer part of a in A. See also 1300 and 1301.	$a' =$ int pt a	QE	4
1706	Set $a' = +1$, 0 or -1 as s >, =, or < zero.	$a' =$ sign s	Q	5-6
1707	Set $a' = +1$, 0 or -1 as a >, =, or < zero.	$a' =$ sign a	Q	4-5
1713	Raise aq to the power s and place the result in am, provided that $aq \geq 0$, Fault if $aq < 0$.	$am' = aq^s$ $aq \geq 0$	QRE	
1714	Place the reciprocal of s in Am.	$am' = 1/s$	QREDO	4
1715	Place the reciprocal of am in Am.	$am' = 1/am$	QREDO	4
1754	Round am by R+, clear L and standardise.	$am' = a$, $l' = 0$	QR+	6
1756	Interchange the contents of S and Am (with no standardising)	$am' = s$, $s' = am$		8
1757	Place the result of dividing s by am in Am.	$am' = s/am$	QREDO	4
1760	Square the contents of Am	$am' = am^2$	QRE	3
1774	Divide am by s and place the result in Am. The original numbers need not be standardised.	$am' = am/s$	QREDO	10
1775	Divide aq by s and place the result in Am. The original numbers need not be standardised.	$am' = aq/s$	QREDO	9

1774 and 1775, besides providing a division instruction which operates on unstandardised numbers, store information which enables extracodes 1776 and 1407 to calculate a quotient and remainder.

1776	When used after division extracodes 1774, 1775, 1574 or 1575, with no other extracodes in between and am unaltered, the definition of 1776 is as follows: Place the quotient of the previous division in s and the remainder in Am, where the remainder has the sign of the divisor.	$s' =$ quotient $am' =$ remainder	QREDO	13
1407	As 1776 except that the quotient is integral and is adjusted according to the sign of the remainder, which is specified by Ba as follows:	$s' =$ adjusted integral quotient $am' =$ remainder	QEDO	

Ba	Sign of remainder
0	Same as the denominator
1	Opposite to the denominator
2	Same as the numerator
3	Opposite to the numerator
4	Same as the quotient
5	Opposite to the quotient
6	Positive
7	Negative

1467	Evaluate the polynomial $s_0 + s_1.an + s_2.an^2 + \ldots s_{ba}.an^{ba}$ where s_0 is the number at S, s_1 at S + 1, etc. and the order of the polynomial is given as an integer in Ba.	$am' = \sum_{r=0}^{ba} s_r.am^r$ where $S_r = S + r$	QRE	6+3ba
1466	Multiply the two numbers at addresses (N + ba + bm) and (N + bm) and add the double-length result into the full accumulator. Rounding takes place near the least-significant end of L. (In detail, when the double-length product has been formed, its least-significant half is first added in M to the least-significant half of the original contents of A. This addition is rounded. The rest of the product and the original contents of M are then added into A without rounding).	$a' = a + C(N+bm) \times C(N+ba+bm)$	QE	18

1415 Generate pseudo-random numbers (PRN's) in A and S (or S*) from numbers in S and S*. This extracode may be used in several ways.

1. With digit 21 of S equal to 0, the PRN is placed in S and A.
   (a) If $s^*y = 0$, $sx > 0$ and $s^*x > 0$, then s' will be a PRN in the range 0 to 8^{sy}, rectangularly distributed and fixed-point (i.e. sx' is a fixed-point PRN and $sy' = sy$). a' will be a PRN in the range 0 to $s^*x.8^{sy}$ (with $al' = s'$).
   (b) If $s^*y = 0$, $sx < 0$ and $s^*x > 0$, then as (a) except that ranges become -8^{sy} to 0 and $-s^*x.8^{sy-1}$ to 0 respectively.
   (c) If $s^*y = 0$ and $s^*x < 0$, then as (a) except that the PRN's alternate in sign.
2. With digit 21 of S = 1, the PRN's are generated in S* and A instead of S and A. The cases are as for 1, interchanging S and S* throughout.

3. Two successive uses of the extracode, with digit 21 of S first = 0 and then = 1, and with $s_y = s^*_y = 0$, will set PRN's in S and S* , both rectangularly distributed in the range 0 to 1. A will contain the product of two PRN's and so will be distributed in the range 0 to 1 with the probability $-\log x\,dx$ of being in the neighbourhood dx of x.

In all cases the generation process must be started with S_x and S^*_x containing numbers with a random mixture of binary digits, but with their least-significant bits set to 1.

7.4.3 Accumulator functions suitable for Fixed-Point Working

1752	Shift a_x up 12 octal places and subtract 12 from a_y.	$m' = a_x \cdot 8^{-12}$ $a_y' = a_y - 12$	A0	10
1753	Shift m down 12 octal places in a_x and increase a_y by 12.	$a_x' = m \cdot 8^{-12}$ $a_y' = a_y + 12$	A0	6
1755	Force a_y to the number n_y given in bits 0-8 of n, shifting a_x up or down accordingly.	$a_x' = a_x \cdot 8^{a_y - n_y}$ $a_y' = n_y$	A0	17
1762	Shift a_x up 12 octal places leaving a_y unchanged.	$m' = a_x \cdot 8^{12}$ $a_y' = a_y$	A0	9
1763	Shift m down 12 places in a_x, leaving a_y unchanged.	$a_x' = m_c 8^{-12}$ $a_y' = a_y$	A0	5
1764	Shift a_x up n octal places, leaving a_y unchanged. If n is negative, shift a_x in the opposite direction.	$a_x' = a_x \cdot 8^{n}$ $a_y' = a_y$	A0	17
1765	Shift a_x down n octal places, leaving a_y unchanged. If n is negative shift a_x in the opposite direction.	$a_x' = a_x \cdot 8^{-n}$ $a_y' = a_y$	A0	12
1766	Place the modulus of s in Am, without standardising. Accumulator overflow will occur if s is -1.0.	$am' = \lvert s \rvert$	A0	4
1767	Place the modulus of am in Am without standardising. A0 will occur if am is -1.0.	$am' = \lvert am \rvert$	A0	3
1772	Multiply m by s_x, shifting the result up by 12 octal places to be in M, and subtracting 12 from a_y.	$m' = (m \cdot s_x)8^{12}$ $a_y' = a_y + s_y - 12$	A0	11
1773	Divide a by s, and force a_y equal to 12, shifting the result, which is in M, if necessary.	$m' = (a_x/s_x) \cdot 8^{a_y - s_y - 12}$ $a_y' = 12$	A0	27
1452	Multiply am by s, forming the answer in A_x. Force a_y to the number given in digits 0-8 of ba, and shift a_x accordingly.	$a_x' = m \cdot s_x \cdot 8^{a_y + s_y - b_{a_y}}$ $a_y' = b_{a_y}$	A0	19-23
1473	Divide a_x by s_x, forming the answer in A_x. Force a_y to the number given in digits 0-8 of ba, and shift a_x accordingly.	$a_x' = (a_x/s_x) \cdot 8^{a_y - s_y - b_{a_y}}$ $a_y' = b_{a_y}$	A0	24-28

Fixed-Point Divisions with Remainder

The three extracodes 1474, 1475 and 1476 each divide some part of the accumulator by the contents of store location S, placing an unstandardised quotient q in the location whose address is ba and leaving an unstandardised remainder r in Am. In all cases, r retains the original sign of am and has a mantissa in the range $0 \le |r_x| < |s_x|$. The quotient is rounded towards zero. Division overflow is set if $s_x = 0$ or -1.0 or if $|s_x| \le$ | mantissa of dividend |. Both D0 and A0 are set when the mantissa of the dividend is equal to -1.0.

If only the remainder is required, one can avoid the need to set ba by putting Ba = B126 in the extracode instruction.

1474	Divide am by s. The exponents of q and r are given by $q_y = a_y - s_y$ and $r_y = a_y - 13$.	C(ba)' = quotient (am/s) am' = remainder (am/s)	D0 A0 E	20-29
1475	Divide a by s. The exponents of q and r are given by $q_y = a_y - s_y$ and $r_y = a_y - 13$	C(ba)' = quotient (a/s) am' = remainder (a/s)	D0 A0 E	19-28
1476	Divide the integral part of am by s. The exponents of q and r are forced to $q_y = 24 - s_y$ and $r_y = 12$. The condition $\lvert am \rvert < 8^{24} \lvert s_x \rvert$ must be observed, otherwise division overflow will occur and the results will be meaningless. The least-significant octal digit of q is always zero, and it is intended that usually $s_y = 12$ so that $q_y = 12$ also and one is working with integers. (In the case $a_y \le -6$ and $am < 0$, this extracode must be preceded by 217, 124, 124, 0 to ensure the true integral part is used).	$C(ba)' = \left(\frac{\text{q int pt am}}{s}\right)$ $am' = r\left(\frac{\text{int pt am}}{s}\right)$	A0 D0 E	28-37

7.4.4 Double-Length Arithmetic

The double-length number s: is stored in two consecutive locations s and s + 1 as two standardised floating-point numbers, where $s_y - 13 \ge s^*_y$. s^* and al are assumed to be always positive. All arithmetic is standardised, rounded and checked for exponent overflow.

1500	Add s: to a	a' = a + s:	10
1501	Subtract s: from a	a' = a - s	10
1502	Negate a and add s:	a' = -a + s:	14
1504	Copy s: into a	a' = s:	4
1505	Copy s: negatively into a	a' = -s:	3

1542	Multiply a by s:	$a' = a.s:$	15
1543	Multiply a negatively by s:	$a' = -a.s:$	19
1556	Store a at S:	$s:' = a$	5
1565	Negate a	$a' = -a$	5
1566	Form the modulus of a	$a' = \|a\|$	4-6
1567	Copy the modulus of s: into A.	$a' = \|s:\|$	5
1576	Divide a by s:	$a' = a/s:$	19

7.4.5 Arithmetic Using the Address as an Operand

The modified address is taken as a 21-bit integer with an octal fraction. Fixed-point operations imply an exponent of 12.

1441	Store ba in S as a fixed-point number	$sx' = ba$, $sy' = 12$		5
1520	Add n to am	$am' = am + n$	QRE	10
1521	Subtract n from am	$am' = am - n$	QRE	9
1524	Place n into a	$a' = n$	Q	8
1525	Place n negatively into a	$a' = -n$	Q	7
1534	Place n into a, without standardising.	$a' = n$		10
1535	Place n negatively into a, without standardising	$a' = -n$		9
1562	Multiply am by n	$am' = am.n$	QRE	8
1574	Divide am by n	$am' = am/n$	QRE	16
1575	Divide aq by n	$am' = aq/n$	QRE	15

After 1574 and 1575, the extracodes 1776 and 1407 can be used to give a remainder and adjusted integral quotient. See section 7.4.2.

7.4.6 Complex Arithmetic

The "complex accumulator" Ca is taken as a pair of consecutive registers, the address of the first one given by the contents of Ba in the instruction. If Ba is B0, Ca will be locations 0 and 1. As with the double-length arithmetic, s: is a number pair consisting of the two numbers at addresses S and S + 1. For Ca and S:, the real part of the number is in the first location, the imaginary part in the second. Ca may coincide with S: if desired, but the two must not partially overlap, i.e. the difference between ba and S must not equal 1. The accumulator is used for the arithmetic so its original contents on entry are spoiled. All arithmetic is standardised, rounded and checked for exponent overflow.

1400	Place the logarithm of s: in Ca	$ca' = \log s:$	
1402	Place the exponential of s: in Ca	$ca' = \exp s:$	140
1403	Place the conjugate of s: in Ca	$ca' = \text{conj } s:$	5
1410	Place the square root of s: in Ca	$ca: = +\sqrt{s:}$	≤117

1411	Place the argument of s: (radians) in Am.	$am' = \arg s:$	
1412	Place the modulus of s: in Am.	$am' = \text{mod } s:$	≤53
1413	Form the numbers s cos s*, s sin s* and place these in Ca. (s* is in radians).	$ca' = s.\cos s^*$, $s.\sin s^*$	95
1414	Place the reciprocal of s: in Ca.	$ca' = 1/s:$	15
1420	Add s: to ca	$ca' = ca + s:$	8
1421	Subtract s: from ca	$ca' = ca - s:$	8
1424	Copy s: into Ca	$ca' = s:$	6
1425	Copy s: negatively into Ca	$ca' = -s:$	6
1456	Copy ca into S:	$s:' = ca$	5
1462	Multiply ca by s:	$ca' = ca.s:$	18

Note: 1400 - the imaginary part of the complex logarithm will lie in the range $-\pi$ (not inclusive) to π (inclusive).
1410 - of the two possible values of the complex square root, the one computed here has a non-negative real part; the remaining ambiguity about the square roots of negative real numbers is removed by computing the one whose imaginary part is positive.

7.4.7 Vector Arithmetic

The following instructions operate on two vectors s_1 and s_2. Both vectors consist of lists of floating-point numbers stored in successive locations. In each instruction the singly-modified address n gives the number of terms in the vectors (i.e. the order) and Ba gives the starting address of s_1. The next B-register after Ba, Ba*, gives the starting address of s_2. Address n must be a positive integer.

Besides their uses in vector and matrix arithmetic, these instructions can be used to manipulate lists of numbers in the store.

The accumulator is used in the arithmetic so its original contents on entry are lost. All operations are standardised rounded and checked for exponent overflow.

1430	Add the vector s_2, which consists of n successive numbers starting at C(ba*) into the vector s_1, which consists of n successive numbers starting at C(ba).	$s_1' = s_1 + s_2$	9 + 4n
1431	Subtract s_2 from s_1.	$s_1' = s_1 - s_2$	9 + 4n
1432	Multiply each term of s_2 by am and store the resultant vector at s_1.	$s_1' = am.s_2$	10 + 4n
1433	Multiply s_2 by am and add this to s_1.	$s_1' = s_1 + am.s_2$	10 + 5n

1434	Copy s_2 to s_1	$s_1' = s_2$		13 + 3n
1436	Form in Am the scalar product: $s_{10} \cdot s_{20} + \cdots + s_{1(n-1)} \cdot s_{2(n-1)}$, where $s_{10}, s_{11}, s_{12}, \cdots$, $s_{1(n-1)}$ are the numbers in S_1, and $s_{20}, s_{21}, \cdots$ are the numbers in s_2.	$am' = \sum_{i=0}^{n-1} s_{1i} \cdot s_{2i}$		
1437	As 1436 but forming the scalar product to double-length accuracy in a.	$a' = \sum_{i=0}^{n-1} s_{1i} \cdot s_{2i}$		10 + 13n

7.4.8 Half-Word Packing

Half-word floating-point numbers consisting of 8-bit exponents and 16-bit mantissae are sometimes useful for low-accuracy calculations where it is necessary to reduce store usage.

1624	Transfer the floating-point number at S into the accumulator, without standardising.	$a' = s$		6
1626	Copy a_y and the 16 most-significant digits of a_x into S after rounding this number in Am by forcing a one in its lowest bit if the rest of a_x is non-zero.	$s' = am$	R	8

7.5 B-Register Arithmetic

7.5.1 General B-Register Operations

1300	Place in Ba the integral part of the floating-point number s. Place the fractional part in Am.	$ba' =$ int pt of s $am' =$ frac pt of s	
1301	Place in Ba the integral part of am. Place the fractional part in Am.	$ba' =$ int pt of am $am' =$ frac pt of am.	

The following six instructions provide integer multiplication and division of ba by n.

For 1302 - 1304, ba and n are interpreted in the normal way as 21-bit integers with a least-significant octal fraction. In the multiplication instructions octal fractions are rounded away from zero, and overflow of the answer is not detected. The accumulator is used in the calculation, but am is preserved.

1302	Multiply ba by n and place the result in Ba.	$ba' = ba \times n$	23-24
1303	Multiply ba negatively by n and place the result in Ba.	$ba' = -ba \times n$	22-23
1304	Divide ba by n. Place the integer quotient in Ba and the remainder, which has the sign of the dividend, in B97.	$ba' =$ int pt (ba/n) $b97' =$ remainder	25-28

For 1312 - 1314, ba and n are interpreted as 24-bit integers, and the result is again a 24-bit integer.

1312	Multiply ba by n and place the result in Ba.	$ba' = ba \times n$	23-24
1313	Multiply ba negatively by n and place the result in Ba.	$ba' = -ba \times n$	22-23
1314	Divide ba by n. Place the integer quotient at the least-significant end of Ba and the remainder, which has the sign of the dividend, as a 24-bit integer in B97.	$ba' =$ int pt (ba/n) $b97' =$ remainder	

The following six instructions provide general n-place shifts of numbers in B-registers.

In arithmetic shifts, the sign digit is propagated at the most-significant end of the register for shifts to the right (i.e. down).

In logical shifts the sign digit is not propagated.

For both arithmetic and logical shifts the result is unrounded on shifts down. In circular shifts, digits shifted off the most-significant end of the register reappear at the least-significant end and vice-versa. n is an integer in bits 0-20 as usual, with no octal fraction. (If n has an octal fraction the answer may be wrong by a shift of one place). In each case, if n is negative a shift of n places in the opposite direction occurs.

1340	Shift ba arithmetically to the right by n places.	$ba' = ba.2^{-n}$	10-22
1341	Shift ba arithmetically to to the left by n places.	$ba' = ba.2^{n}$	9-21
1342	Shift ba circularly to the right by n places.	$ba' = ba.2^{-n}$, circular shift	10-19
1343	Shift ba circularly to the left by n places.	$ba' = ba.2^{n}$, circular shift	9-18
1344	Shift ba logically to the right by n places.		10-21
1345	Shift ba logically to the left by n places.		9-20

The following are miscellaneous arithmetic instructions on half-words and index registers.

1347	Perform the logical "OR" operation on ba and s and place the result in S.	$s' = ba \vee s$	5
1353	Set B123 by writing n to it, and read the result to Ba. This sets ba equal to the position of the most-significant 1 bit in bits 16-23 of n. (B123 is described in Chapter 4.)	$b123' = n$, then $ba' = b123$	
1356	Set the B-test register as the result of non-equivalencing ba and s.	$bt' = ba \not\equiv s$	7
1357	Set Bt as the result of non-equivalencing ba and n.	$bt' = ba \not\equiv n$	5
1376	Set Bt as the result of collating ba and s.	$bt' = ba \,\&\, s$	5
1377	Set Bt as the result of collating ba and n.	$bt' = ba \,\&\, n$	
1364	Preserve the digits of Ba where there are zeros in n and copy digits from Bm into Ba where there are ones in n.	$ba' = (ba \,\&\, \bar{n}) \vee (bm \,\&\, n)$ [also $b119' = (ba \not\equiv bm) \,\&\, n$]	4
1371	Dummy extracode to set up b121 and b119.	$b121' = Ba$, $b119' = N + bm$.	

1771	Dummy extracode to set up b121 and b119.	$b121' = Ba$, $b119' = N + ba + bm$.	

7.5.2 Character Data Processing

1131 Search for s in table starting at C(ba). If s can be found, ba' will record its address, otherwise the sign bit of ba' will be set to 1. Main control is re-entered at c' = c + 2, and C(c + 1) is used to specify parameters k, l, m as shown below. Up to l + 1 half-words are scanned, starting with C(ba) and continuing at intervals of k half-words, each being masked with m before comparison with s.

bits	0-9	10-20	21-23	0-23
	k	l	spare	m
	interval	count		mask

In the following two instructions S is taken as a character address, the octal fraction giving the address of the 6-bit character within the word.

1250	Place the character s in the least-significant 6 bits of Ba and clear the other digits of Ba.	$ba' = char\ s$	7-10
1251	Copy the character from the least-significant 6-bits of Ba into the character position at S, leaving the other characters in the word unaltered.	$s' = char\ ba$	11-18

In the following two instructions ba is interpreted as a character address, and the content of the next B-register, ba*, is interpreted as a half-word address. n is used as a count and its octal fraction must be zero.

1252	Unpack n characters. The n characters, packed in successive character positions starting at C(ba), are placed in the least-significant 6-bits of n successive half-words starting at C(ba*). The other digits in each half-word are set to zero.		16 + int pt ($6\frac{3}{4}$n)
1253	Pack n characters. Take the n characters stored in the least-significant 6-bits of n successive half-words starting at C(ba*) and pack these into n successive character positions starting at C(ba).		18 + 5n

7.5.3 Logical Accumulator Instructions.

B98 and B99 are used in these instructions as a double-length B-register. This is called the logical accumulator and denoted by G.

1204	Starting at the most-significant end, count the number of 6-bit characters which are identical in g and s, continuing only until the first dissimilar characters are found. Place the result in Ba.		10-31
1265	Shift g up by 6 places, writing overspill to Ba, and add n.	ba′ = m.s. character of g. $g' = 2^6 g + n$	11
1601	Copy s into G.	$g' = s$	3
1604	Add s into G.	$g' = g + s$	7
1605	Add s into G, adding any overflow carry in again at the least-significant end.	$g' = g + s$ with end-around carry	12
1606	Non-equivalence s with g	$g' = g \not\equiv s$	4
1607	Collate s with g	$g' = g \,\&\, s$	3
1611	Replace g by its logical binary complement.	$g' = \bar{g}$	3
1613	Copy g into S	$s' = g$	3
1615	Copy g into Am, without standardising.	$am' = g$	4
1630	Form the logical binary complement of s and collate this with g.	$g' = g \,\&\, \bar{s}$	5
1635	Copy am into G.	$g' = am$	4
1646	"OR" s with g	$g' = g \vee s$	3
1652	Set Bt by the result of subtracting s from g.	$bt' = g - s$	7-9

7.6 Test Instructions

7.6.1 Accumulator Test Instructions

1200	Place n in Ba if the Accumulator overflow (A0) is set. Clear A0.	ba′ = n if A0 is set.	9
1201	Place n in Ba if A0 is not set. Clear A0.	ba′ = n if A0 is not set.	7
1234	Increase main control by 2 (instead of by 1) if am is approximately equal to s.	$c' = c + 2$ if $am \sim s$	11
1235	Increase main control by 2 if am is not approximately equal to s.	$c' = c + 2$ if $am \not\sim s$	11

For 1234 and 1235, approximate equality is defined as

$$\left| \frac{am - s}{am} \right| < C(ba)$$

am must be standardised on entry. By definition, if am = 0 then am is not approximately equal to s.

1236	Place n in Ba if am is greater than zero.	ba′ = n if am > 0	4-6
1237	Place n in Ba if am is less than or equal to zero.	$am \leq 0$	3-5
1255	Place n in Ba if m is neither zero nor all ones.	ba′ = n if m ≠ all 1's or all 0's	
1727	Depending on whether am is greater than, equal to, or less than s, increase main control by 1, 2 or 3.	c′ = c + 1, 2, 3 as am >, =, < s	7
1736	Increase main control by 2 if the modulus of am is greater than or equal to s.	$c' = c + 2$ if $\lvert am \rvert \geq s$	
1737	Increase main control by 2 if the modulus of am is less than s.	$c' = c + 2$ if $\lvert am \rvert < s$	

In 1234, 1235, 1727, 1736 and 1737 am is preserved but l is not.

7.6.2 B-register Test Instructions

1206	Place n in Ba if the most significant 6-bit character in G is zero.		4