User Hardware Handbook – Computer

CENTRAL PROCESSOR UNIT INSTRUCTION SET

The information presented herein is, to the best of our knowledge, true and accurate. No warranty or guarantee, expressed or implied, is made regarding the accuracy of information supplied or capacity, performance or suitability of any product or service since the manner of use is beyond our control.

You are advised that you should ensure that the information contained herein has not been superseded.

All our products, materials and services are sold subject to our Conditions of Sale, available on request.

GEC COMPUTERS LIMITED
Elstree Way, Borehamwood, Hertfordshire.
Telephone No. 01-953-2030

Holding Company – the General Electric Company Limited of England

CPU INSTRUCTION SET

CONTENTS

Page

1. INTRODUCTION

1.1 Notation Used 1

2. PROGRAM ACCESSIBLE REGISTERS 2

3. OPERANDS

3.1 Operand Lengths 3

3.2 Operand Addresses 3

4. INSTRUCTION FORMATS

4.1 Format A 4

4.2 Format L 8

4.3 Format B 8

4.4 Format RR 9

5. MODES OF OPERATION

5.1 Basic Test 11

5.2 Full Nucleus 11

5.3 Fixed Point Mode 11

5.4 Floating Point Mode 11

5.5 Mode Control 11

6. FIXED POINT OPERATIONS

6.1 Number Representation 12

6.2 Mixed Length Operation 12

6.3 Condition Markers 12

6.4 Logical Operations 14

Page

7. THE FIXED POINT INSTRUCTION SET: FORMATS A1–A5

7.1 Operations on the 32 Bit Accumulator 15

7.2 Operations on Operands in Store 21

7.3 Operations on the X Register 23

7.4 Operations on Y and Z Registers 26

7.5 Multiple Load and Store Instructions 29

7.6 Indirect Branches 30

8. INSTRUCTIONS AVAILABLE IN FORMAT B 32

9. INSTRUCTIONS AVAILABLE IN FORMAT L

9.1 Operations on the 32 Bit Accumulator 33

9.2 Literal Operations on the X Register 35

9.3 Literal Operations on the Y and Z Registers 38

9.4 Condition Branch Instructions in Format L 40

9.5 Shift Instructions 43

9.6 Bit Manipulation Instructions 47

9.7 String Manipulation 50

9.8 Miscellaneous Instructions in Format L 53

10. INSTRUCTIONS AVAILABLE IN FORMAT RR 56

11. FLOATING POINT OPERATION

11.1 Floating Point Number Representation 61

11.2 Floating Point Store Format 61

11.3 Floating Point Register Format 61

11.4 Normal Representation 62

11.5 Floating Point Zero 62

11.6 Mixed Precision Operation 62

11.7 Overflow and Underflow 63

Page

12. THE FLOATING POINT INSTRUCTION SET

12.1 Normal Length Operations 64

12.2 Extended Operations 66

12.3 Floating Point Conversion Instructions 69

12.4 Floating Point Instructions in Format L 69

Appendix 12A: Floating Point Arithmetic 71

13. CONTROL INSTRUCTIONS AVAILABLE IN BASIC MODE 74

SUPPLEMENT

Supplement 1: Summary of Instructions

1. INTRODUCTION

This manual describes the instruction set of the GEC 4000 Series computers, except for Nucleus instructions CALL, ICB, SEM and SEG described in detail in CPU NUCLEUS MANUAL.

Excluding the above Nucleus instructions the instruction repertoire of the computers contains some 155 different instructions. Certain of these instructions may be specified in 1 of 5 operand addressing formats.

The processor is capable of performing both integer and floating point arithmetic and instructions are provided for both modes of operation.

1.1 NOTATION USED

In this manual, the following notation is used to describe the instruction set.

a denotes the content of the 32 bit Accumulator A.
da denotes the content of the 64 bit Extended Accumulator BA.
ha denotes the content of the least significant 16 bits of A.
ba denotes the content of the least significant 8 bits of A.
b denotes the content of the 32 bit Accumulator Extention B.
x denotes the content of the 16 bit Index register X.
y denotes the content of the 16 bit Y register.
z denotes the content of the 16 bit Z register.
s denotes the content of the 16 bit Sequence Register S.
l denotes the content of the 16 bit Local Workspace Register L.
fa denotes the short floating point number in the Floating Accumulator.
ea denotes the long floating point number in the Floating Accumulator.
Q denotes the operand address generated by an instruction.
wq denotes the fullword store operand of an instruction.
hq denotes the halfword store operand of an instruction.
bq denotes the byte store operand of an instruction.
fq denotes the short Floating Point operand of an instruction.
eq denotes the long Floating Point operand of an instruction.

2. PROGRAM ACCESSIBLE REGISTERS

The following program accessible registers are provided.

32 bit Accumulator, A

The accumulator is divided into two 16 bit registers AM and AL. This accumulator is used to hold the result of fixed point arithmetic and logical operations. It is also used to hold the least significant 32 bits of the mantissa in the case of long floating point operations.

32 bit Accumulator Extension, B

This register is divided into two 16 bit registers BM and BL and is used to hold remainders in integer divide instructions and the mantissa of the result in normal length floating point operations.

BM and BL may be used in conjunction with AM and AL to provide a 64 bit register and in this case the most significant register is BM and the least significant is AL. This 64 bit register is used to hold products in fixed point multiply instructions and to hold the mantissa of the result in extended-length, floating-point operations.

16 bit X Register

The X register is used as an index register to address array elements in store. It may also be used as a secondary accumulator and a comprehensive set of instructions is provided for this purpose.

16 bit Y and Z Registers

Both Y and Z registers are used to hold the base address of areas of data such as records. A restricted instruction repertoire is provided for operations on Y and Z.

16 bit L Register

This is a local workspace pointer register and holds the base address of the area of store containing the local workspace of a program. The L register is operated on by Nucleus branch instructions and by instructions using data held in other registers.

16 bit S Register

This is the sequence control register and normally contains the address of the next instruction in sequence. All instructions are 16 bit halfwords and this register is incremented by 2 for each instruction executed. The S register is operated on by branch instructions and by instructions using data held in other registers.

8 bit C Register

This is the control register and contains various flags that may be set by a program at any time.

These flags are referred to as 'Condition Markers' and are fully described in section 6.

The register holds condition markers as follows:—

0							7
0	0	0	FM	N	Z	OF	CA

FM	-	FLOATING MARKER
N	-	NEGATIVE CONDITION MARKER
Z	-	ZERO CONDITION MARKER
OF	-	OVERFLOW CONDITION MARKER
CA	-	CARRY CONDITION MARKER

3. OPERANDS

3.1 OPERAND LENGTHS

Information is manipulated in multiples of eight bits. Each 8 bit unit of information is called a Byte.

Bytes may be handled separately or grouped together as follows.

(a) *Halfwords*

A Halfword comprises two consecutive bytes. The low addressed byte of the item must be held at a byte address divisible by two in main store. Instructions and single precision integer operands are held as Halfwords.

(b) *Fullword*

A Fullword comprises 4 consecutive bytes. The low addressed byte of the item must be held at a byte address divisible by 4 in main store. Double length integer operands and short Floating Point operands are held as Fullwords.

(c) *Double Word*

A Double word comprises 8 consecutive bytes. The low addressed byte of the item must be held at a byte address divisible by 8 in main store. Long Floating Point operands are held as Double words.

3.2 OPERAND ADDRESSES

Each store reference instruction specifies directly or indirectly the required operand address and the type of operand i.e. Byte, Halfword, Fullword or Double Word. The operand addresses are formed as described in section 4 and the hardware ensures that an operand address of the correct form is presented to the main store at every access. i.e. If a Halfword is requested from store the least significant bit will be forced to zero. Similarly, for other store addresses:—

TYPE OF STORE ACCESS	LEAST SIGNIFICANT 3 STORE ADDRESS BITS
Byte	X X X
Halfword	X X 0
Fullword	X 0 0
Double Word	0 0 0

4. INSTRUCTION FORMATS

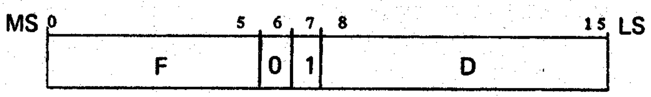
Eight instruction formats are provided, known as formats A1, A2, A3, A4, A5, B, RR and L. The 16 bits of an instruction are divided into several fields which together specify the operation to be performed and where necessary the operand address.

Formats A1 to A5 are referred to as format A instructions and have a common instruction set. Each format provides a different method of forming the operand address. A separate set of instructions is available in each of the formats B, RR and L.

4.1 FORMAT A

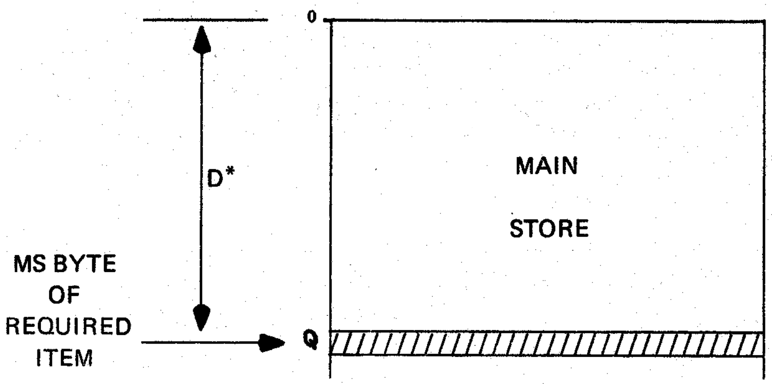
Instructions specified in this format are used for arithmetic, and logical operations. Some instructions in this format may be performed in either integer mode or floating mode under control of the FM flag (section 6).

Format A1



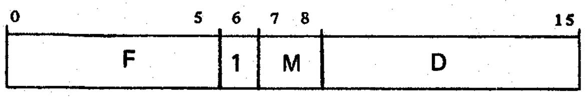
Of the 16 bits used to define the instruction the most significant 6 bits (0–5) are used to specify one of 64 possible instructions. The next 2 bits (6 and 7) specify the format and the final 8 bits are the displacement field that defines the address.

In this format the displacement is used to form the operand store address, after scaling as described below. D* denotes the scaled displacement.

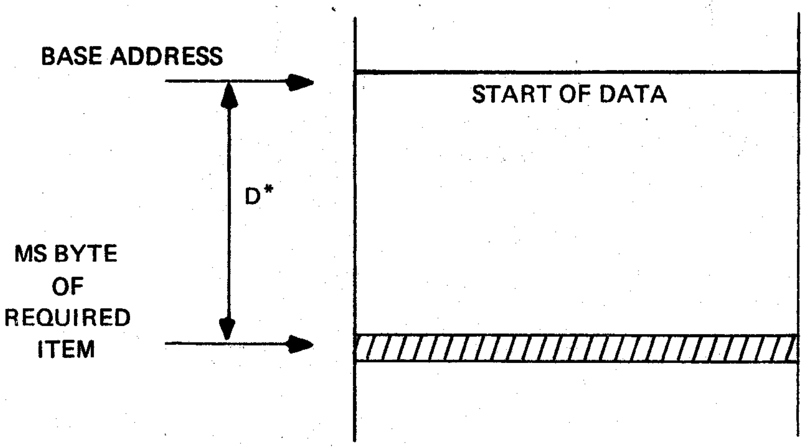


The displacement field is scaled according to operand length. If a byte operand is required no scaling takes place and the eight bits of the displacement field are able to access any of the first 256 bytes of virtual store. An instruction requiring a halfword operand may access 256 halfwords in the first 512 bytes of virtual store. Similarly 256 full words in the first 1024 bytes of virtual store or 256 double words in the first 2048 bytes of virtual store. .This format is used for accessing global simple variables.

Format A2



The most significant 6 bits specify the function as in format A1. Bit 6 specifies the format whilst bits 7 and 8 (M Field) define how the D Field is to be used, to define the operand address. In this format the scaled displacement field is added to a base register specified by the M Field.



The displacement field is scaled as for format A1 and the 7 bits of this field may be used to access one of 128 items (bytes, halfwords, words, or double words) in the area defined by a base register.

Operand addresses are formed as tabulated with the value of the M field selecting base registers L, S, Y or Z.

In the following table D* denotes the scaled displacement field.

M	OPERAND ADDRESS
0	l + D*
1	s + D*
2	y + D*
3	z + D*

The L register normally holds a pointer to the local data needed in a program chapter. Mode 0 is thus used to access local simple variables.

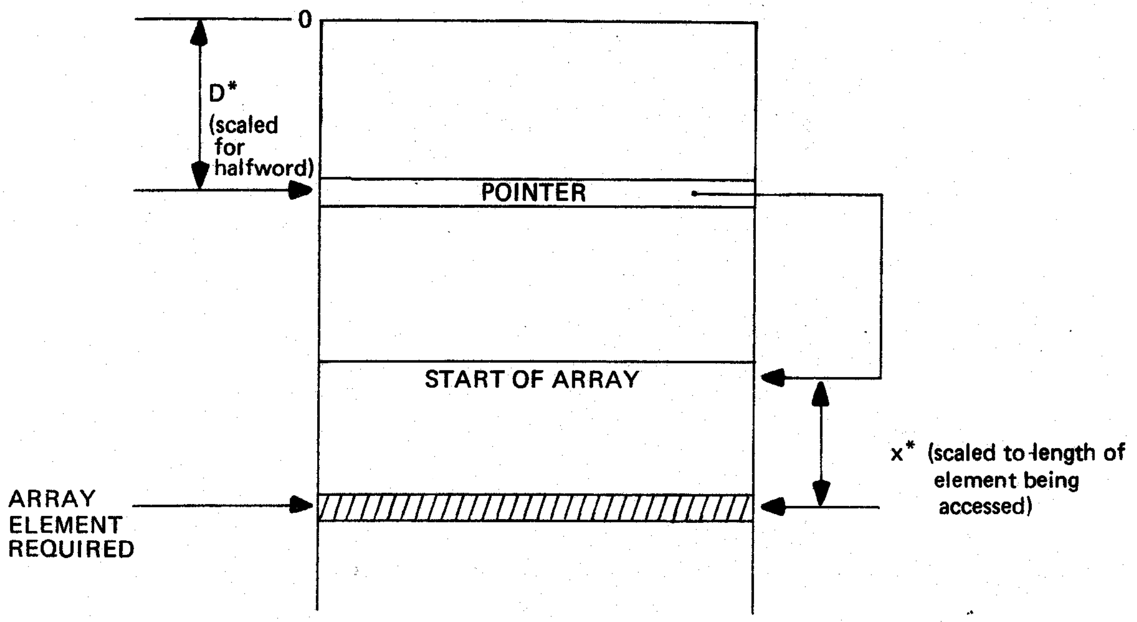
Mode 1 is used to access constants held in the same area of store as the code for a particular program chapter. These constants must be at a higher address in store than the instruction that accesses them.

Registers Y and Z normally hold pointers to records, and therefore modes 2 and 3 are used to access general simple variables.

Format A3

0 5	6	7	8	9	10 15
F	0	0	0	1	D

The most significant six bits are used to specify the function as in format A1. Bits 6,7,8 and 9 specify the format and the final 6 bits are the displacement field that defines the store address. In this format a halfword in store is accessed and used to form the base address of an array. This base address is then indexed to form the address of a selected array element. The index register x is scaled according to the length of the element being accessed.

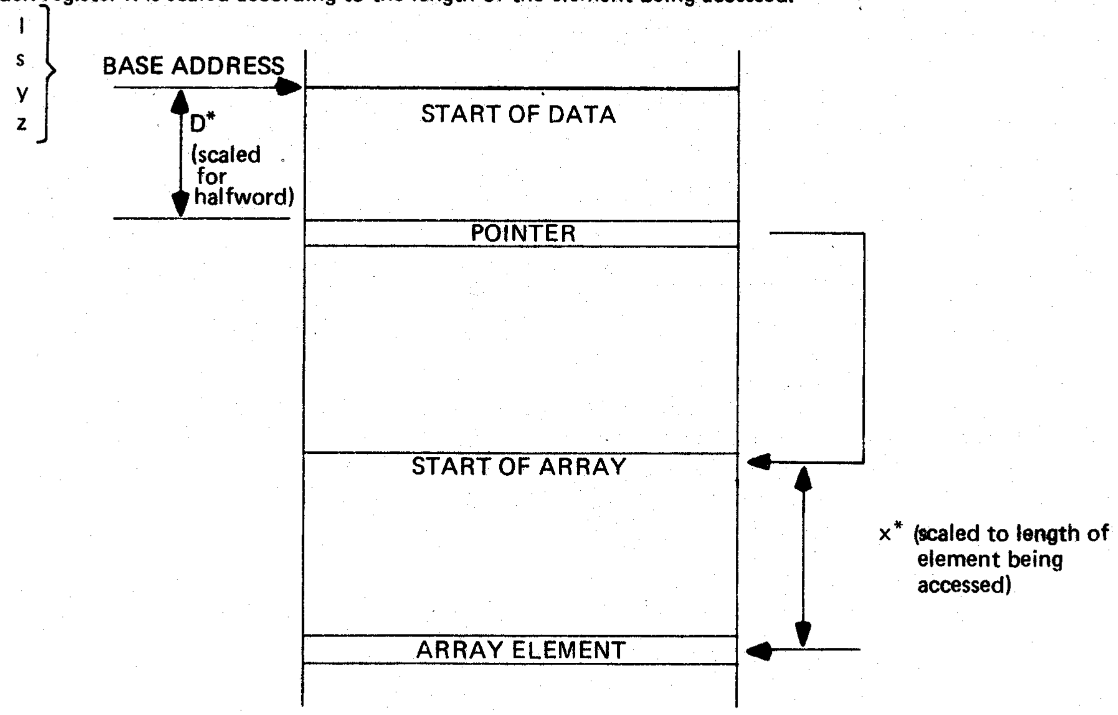


This format is used to access global arrays, the 6 bits of the displacement field allowing up to 64 array pointers to be used.

Format A4

0 5	6	7	8	9 10	11 15
F	0	0	1	M	D

The most significant six bits are used to specify the function as in format A1. Bits 6,7 and 8 specify the format whilst 9 and 10 define the way in which the displacement field is to be used, to form the operand address. In this format the displacement field is always scaled for a halfword and then added to a base register specified by M. The address so formed is used to access a halfword operand from store, which forms the base address of an array. The operand is indexed with x to form the address of an individual array element. The index register x is scaled according to the length of the element being accessed.



The 5 bit displacement field may be used to access one of 32 array pointers in an area of store defined by a base register.

Operand addresses are formed as tabulated with the value of the M field selecting base registers L, S, Y or Z.

In the following table D denotes the value of the displacement field in the instruction and x* denotes the scaled value of the x register.

M	OPERAND ADDRESS
0	(l + 2D) + x*
1	(s + 2D) + x*
2	(y+ 2D)+ x*
3	(z+ 2D)+ x*

Mode 0 is used to access local arrays of data.

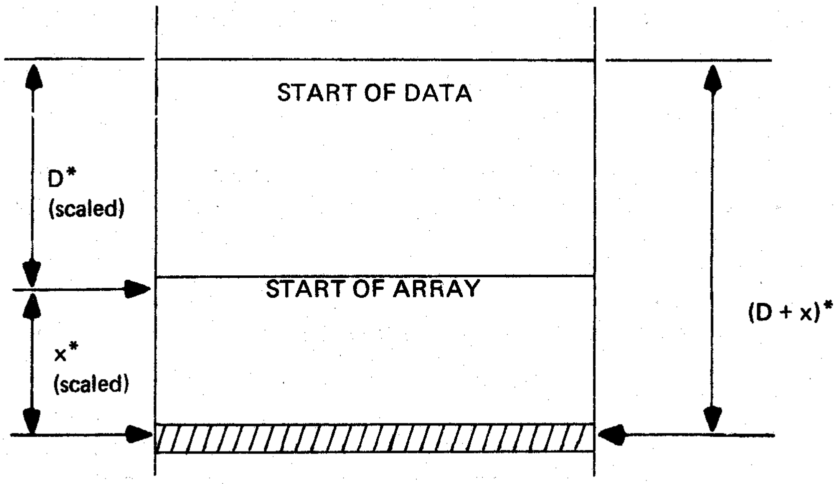
Mode 1 is used to access arrays of constants that are held in the same area of store as the code for a particular program chapter, these constarrts being at higher store addresses than the instructions accessing them.

Modes 2 and 3 are used to access arrays of data held in records.

Format A5

0 5	6	7	8	9	10 11	12 15
F	0	0	0	0	M	D

The most significant six bits are used to specify the function as in format A1. Bits 6,7,8 and 9 specify the format whilst 10 and 11 define the way in which the operand address is formed. This format is similar to the A2 format except that the scaled displacement field is added to a base register and then indexed, before forming the operand address.



The displacement field and x are scaled according to the length of the item to be accessed from store. The four bits of displacement field enable 16 items (bytes halfwords, words or doublewords) to be accessed from store in the area defined by a base register and the index register x.

Operand addresses are formed as tabulated with the value of M field selecting base registers L, Y or Z.

In the following table D^* denotes the scaled displacement field and x^* denotes the scaled value of the x register.

M	OPERAND ADDRESS
0	$I + D^* + x^*$
1	$D^* + x^*$
2	$y + D^* + x^*$
3	$z + D^* + x^*$

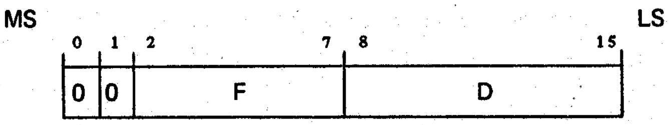
Mode 0 is used to access local arrays of data.

Mode 1 is used to access global arrays of data.

Modes 2 and 3 are used to access arrays of data such as records and vectors.

4.2 FORMAT L

The instructions specified in this format are literal instructions and generally do not require an operand from store.



For this format the two most significant bits (0–1) are always zero. The next 6 bits (2–7) specify the function and the remaining 8 bits are used as follows:–

(a) As an 8 bit unsigned literal operand. Any number between 0 and 255 can be represented.

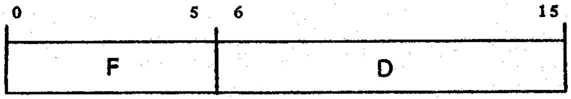
(b) Some conditional branch instructions are specified in this format and here the displacement field is treated as a signed integer, to specify a branch destination. This destination may be within 127 halfwords (i.e. instructions) forward or 128 halfwords backward relative to the next instruction in sequence.

Thus: Operand address = S ± 2D

(c) Where a literal operand is not required and the instruction is not a conditional branch, bits 8–15 are used in conjunction with the F bits to further define the instruction (e.g. shifts and control functions).

4.3 FORMAT B

Two instructions only are specified in this format and both are unconditional branches.



The two instructions in this format are defined by the F bits, the most significant 4 bits (bits 0–3) are always zero. The remaining 10 bits are used as a signed integer to specify the branch destination. This destination address may be within 511 halfwords (i.e. instructions) forwards or 512 halfwords backwards relative to the next instruction in sequence.

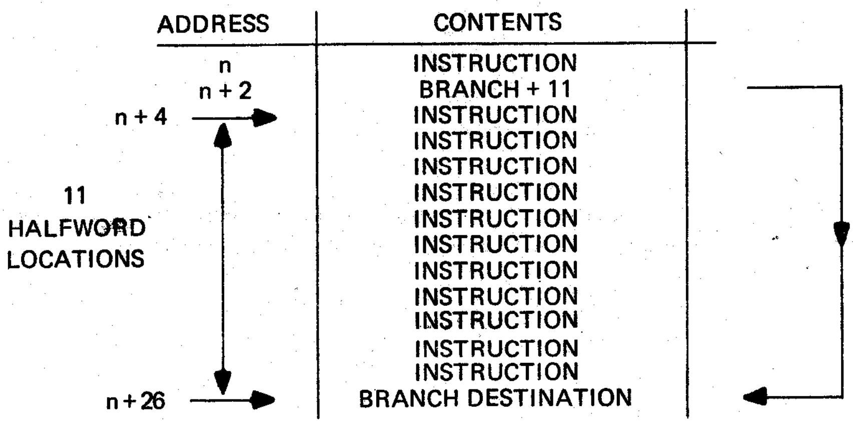
Thus: Destination Address = S ± 2D

Since instructions are always 16 bit halfwords the displacement field in format B instructions is always multiplied by 2. At the start of the execution of any instruction, the sequence control register (S) always points to the next instruction in sequence. Therefore all branches are relative to the next instruction in sequence and not the current instruction.

EXAMPLE 1

INSTRUCTION =	0 0 0 0 0 1	0 0 0 0 0 0 1 0 1 1
=	BRANCH	+ 11

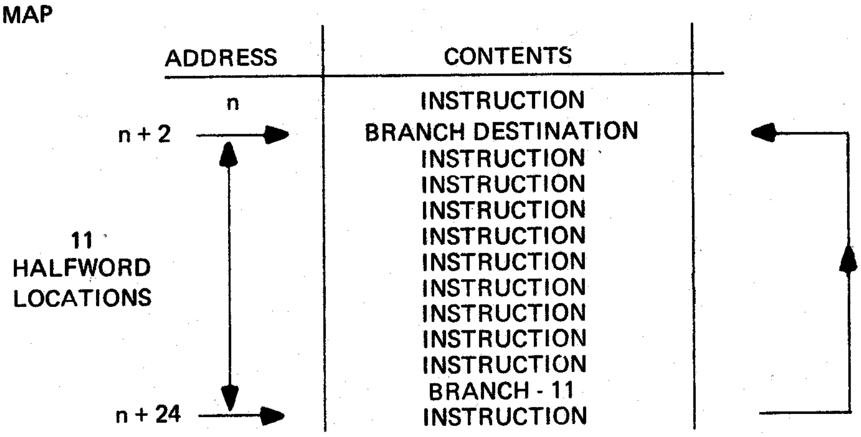
STORE MAP



EXAMPLE 2

INSTRUCTION =	0 0 0 0 0 1	1 1 1 1 1 1 0 1 0 1
=	BRANCH	- 11

STORE MAP



4.4 FORMAT RR

The instructions specified in this format are register to register instructions.

0	1	2	3	4	5	6 9	10 12	13 15
0	0	0	0	0	0	F	G1	G2

The first 6 bits (0–5) are always zero. The next 4 bits (6–9) define 16 possible instructions, which are further defined by bits 10–15.

Format RR instructions are used for operations between registers. The two registers taking part in an operation are defined by G1 and G2. G1 specifies the destination register and G2 specifies the source register as follows:—

G1,G2	REGISTER
0	0 (see below)
1	A (32 bits)
2	B (32 bits)
3	X (16 bits)
4	L (16 bits)
5	S (16 bits)
6	Y (16 bits)
7	Z (16 bits)

Register 0 is a non-existent dummy register. If used as a source it appears to contain zero; if used as a destination the result is lost but the condition markers record the result of the operation.

5. MODES OF OPERATION

The CPU has two different modes of operation, basic test and full nucleus. Within these two modes of operation, two further modes are provided, integer mode and floating point mode.

5.1 BASIC TEST

This mode of operation is provided in order to more easily test the central processor basic instruction set. Most of the microprogram controlling the nucleus is disenabled and the computer becomes a machine capable of running only 2 programs simultaneously. One is the normal operating program, the other being entered on receipt of an interrupt as described in section 13. All addresses are 16 bit absolute addresses and thus 64kBytes of store may be accessed. Certain additional instructions are provided in this mode to facilitate input/output and interrupt handling. These instructions are described in section 13.

Should these instructions be specified when the machine is in full nucleus mode they are treated as undefined instructions (refer to CPU Nucleus Manual).

Further modifications to the functioning of the CPU are:

(1) The action taken when an Input Output Processor or Error interrupt occurs (refer to section 13).

(2) The action taken as a result of depressing certain switches on the front panel (IPL1, IPL2, START/ STOP). This is described in CPU Controls and Monitor Unit Manual.

(3) The 'TRIG C.P.' facility becomes available enabling a program to be restarted at a selected point (see CPU Controls and Monitor Unit Manual).

5.2 FULL NUCLEUS

In this mode, addresses generated by the program are 16 bit virtual addresses and are mapped into absolute addresses by the mechanism described in the CPU Nucleus Manual. Control instructions in this mode are: CALL, SEMAPHORE INTER-CHAPTER BRANCH and SEGMENT and these are fully described in the CPU Nucleus manual.

5.3 FIXED POINT MODE

In this mode of operation the fixed point instruction set described in sections 7 – 10 is provided.

5.4 FLOATING POINT MODE

In this mode of operation certain instructions are treated as floating point instructions as described in section 11.

5.5 MODE CONTROL

The modes of operation described in 5.1 and 5.2 above are controlled by a switch on the CMU front panel (see CMU Document) and under certain conditions by the SFN instruction (see section 9).

The modes of operation described in 5.3 and 5.4 are controlled by the FM flag in the control register which can be manipulated by the program as described in section 6.

6. FIXED POINT OPERATIONS

The fixed point instruction set performs binary arithmetic on operands serving as addresses and index quantities as well as fixed point data. Operands may be 16 bits or 32 bits long and may be held in one of the program accessible registers or in the main store. Both operands are signed 16 or 32 bits long, negative quantities being held in two's complement form. Condition markers are set as a result of most arithmetic and logical operations. Addresses are sometimes treated as positive 16 bit integers.

6.1 NUMBER REPRESENTATION

Fixed point operands are treated as 16 or 32 bit signed binary integers.

Positive integers in the range 0 : 2^{15}- 1 (16 bit) or 2^{31} - 1 (32 bit) are represented directly. The most significant bit of an operand representing a positive integer will therefore be zero.

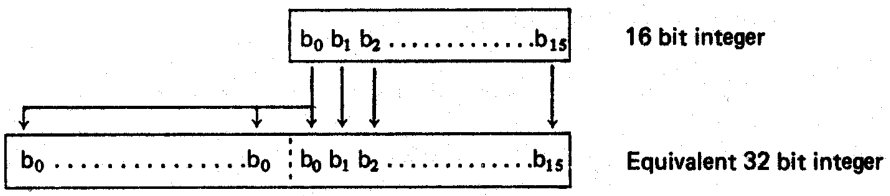
Negative integers in the range -2^{15} : - 1 (16 bit) or - 2^{31} : - 1 (32 bit) are represented by subtracting the magnitude of the number from 2^{16} (16 bit) or 2^{32} (32 bit). The most significant bit of a negative operand will therefore be a one.

Since the most significant operand bit can be used to distinguish the sign of an operand, it is referred to as the sign bit. This representation is known as 2s complement notation. The range of numbers which can be represented is therefore

$$-2^{15} \leqslant N \leqslant 2^{15} - 1 \quad \text{(halfword operands)}$$
$$\text{or} \quad -2^{31} \leqslant N \leqslant 2^{31} - 1 \quad \text{(fullword operands)}$$

6.2 MIXED LENGTH OPERATIONS

A 16 bit 2s complement integer can be converted into the equivalent 32 bit representation as follows:— the least significant 16 bits of the 32 bit integer are the same as the original 16 bit integer, and the sign bit of the 16 bit integer is replicated throughout the most significant 16 bits of the 32 bit integer.


b0 b1 b2 . . . . .b15
16 bit integer
b0 . . . . .b0
b0 b1 b2 . . . . .b15
Equivalent 32 bit integer

This process is referred to as 'sign extending'. It is therefore possible to perform operations between 32 bit and 16 bit integers provided 16 bit operands are sign extended to 32 bits before the operation is performed.

6.3 CONDITION MARKERS

There are four condition markers N, Z, OF and CA which are used to record information about an operation. This information may subsequently be tested with a conditional branch instruction, in considering the information conveyed by these markers it is necessary to distinguish between two results that an operation may produce. The first is the TRUE result, obtained by applying the rules of binary arithmetic to the operation, the second is the APPARENT result obtained by taking the least significant n bits of the true result, where n is the number of bits available for recording the result.

The meanings of the condition markers are as follows:—

The Negative Condition Marker (N)

This is a single bit set to the sign of the true result of the last operation performed. Thus, it is set to a logical 1 if the true result is negative and to a logical 0 if the true result is positive.

The Zero Condition Marker (Z)

This is a single bit set to a logical 1 if the apparent result of the last operation performed is zero, and is reset to logical 0 if the apparent result of the last operation performed is non-zero.

The Overflow Condition Marker (OF)

This is a single bit set to a logical 1 if an operation produced overflow since the flag was last reset i.e. the apparent result is different from the true result. This flag is reset by obeying the instruction 'Branch on Overflow'.

The Carry Condition Marker (CA)

This is a single bit which, following an add operation is set to logic 1 if the operation produced a carry out of the most significant bit position, and is otherwise reset to 0, and following a subtract operation is set to logic 1 if the operation produced a borrow out of the most significant bit position, and is otherwise reset to zero.

Floating Marker

This is a single bit that controls the floating point feature of the computer. When this flag is reset to a logical 0 certain instructions operating on the accumulator are interpreted as Integer Operations. When this flag is set to a logical 1 these instructions are interpreted as Floating Point Operations and a small set of additional instructions are made available.

The flag may be set or cleared by control instructions in format L (see section 9) or by use of the Load Multiple Instruction. A further way to clear the FM flag is by use of one variant of the Call instruction (described in CPU Nucleus Manual).

During input/output these flags are used for special purposes.

Examples of negative, zero and overflow and carry conditions are given below assuming eight bits are available to hold the result of the calculation. The carry out bit is also shown, in parentheses:—

Addition

(a)	+57	=	00111001	Negative reset
	+35	=	00100011	Zero reset
	+92	=	(0)01011100	Overflow not set
				Carry reset
(b)	+57	=	00111001	Negative reset
	-35	=	11011101	Zero reset
	+22	=	(1)00010110	Overflow not set
				Carry set
(c)	-57	=	11000111	Negative set
	-92	=	10100100	Zero reset
	- 149	=	(1)01101011	Overflow set (Out of range.Result < -128).
				Carry set
(d)	+57	=	00111001	Negative reset.
	+92	=	01011100	Zero reset
	+149	=	(0)10010101	Overflow set (neg.result)
				Carry reset

Multiplication (Carry Not Affected)

(a)	+35	=	00100011	Negative reset
	+3	=	00000011	Zero reset
	+105	=	01101001	Overflow not set
(b)	+35	=	00100011	Negative set
	-3	=	11111101	Zero reset
	-105	=	10010111	Overflow not set
(c)	+35	=	00100011	Negative reset
	+ 4	=	00000100	Zero reset
	+140	=	10001100	Overflow set
(d)	+32	=	00100000	Negative reset
	+16	=	00010000	Zero set
	+512	=	10 00000000	Overflow set

6.4 LOGICAL OPERATIONS

The condition markers are used in the same way as for integer arithmetic operations, except that carry and overflow stats are unaffected by logical operations.

7. THE FIXED POINT INSTRUCTION SET: FORMATS A1–A5

The following describes the basic fixed point instruction set of the computer, i.e. those instructions provided when FM is set to logic 0. The Floating point instructions in section 12 are not available.

7.1 OPERATIONS ON THE 32 BIT ACCUMULATOR

The following instructions are available:–

LDB	–	LOAD BYTE
LD	–	LOAD HALFWORD
LDW	–	LOAD WORD
STB	–	STORE BYTE
ST	–	STORE HALFWORD
STW	–	STORE WORD
AD	–	ADD HALFWORD
ADW	–	ADD WORD
SB	–	SUBTRACT HALFWORD
SBW	–	SUBTRACT WORD
CPB	–	COMPARE BYTE
CP	–	COMPARE HALFWORD
CPW	–	COMPARE WORD
N	–	AND HALFWORD
NW	–	AND WORD
M	–	MULTIPLY HALFWORD
MW	–	MULTIPLY WORD
D	–	DIVIDE HALFWORD
DW	–	DIVIDE WORD

LDB

0 5	6 15
0 1 0 0 0 0	ADDR

LOAD BYTE: a := bq. The byte is loaded into the A register from store and is extended to 32 bits by most significant zeros before the operation. This instruction therefore loads the least significant byte of the accumulator and clears the most significant 24 bits. The final value in A will lie in the range 0 to 255.

Condition Markers

N – *is cleared by the instruction*
Z – *is set if the 32 Bit accumulator is zero otherwise it is cleared*
CA – *not affected*
OF – *not affected*

LD

0 5	6 15
1 0 0 0 0 0	ADDR

(FM = 0 only)

LOAD: a := hq. The halfword operand is loaded from store and is sign extended to 32 bits before the operation. The instruction therefore loads the least significant 16 bits (A_L) of the accumulator with the operand and loads the most significant 16 bits (A_M) with the operand extension.

Condition Markers

N – *is set to the sign of the 32 bit accumulator after the load*
Z – *is set if the accumulator is zero after the operation otherwise it is cleared*
CA – *not affected*
OF – *not affected*

LDW

0 5	6 15
1 0 1 0 0 0	ADDR

(FM = 0 only)

LOAD WORD: a := wq. The 32 bit operand is loaded from store.

Condition Markers

N – *is set to the sign of the 32 bit accumulator after the load*
Z – *is set if the accumulator is zero after the operation otherwise it is cleared*
CA – *not affected*
OF – *not affected*

STB

0 5	6 15
0 1 0 1 1 1	ADDR

STORE BYTE: bq := ba. The least significant byte of the 32 bit accumulator is stored.

Condition Markers

N – *is cleared by the instruction*
Z – *is set if the byte to be stored is zero otherwise it is cleared*
CA – *not affected*
OF – *not affected*

ST

0 5	6 15
1 0 0 1 1 1	ADDR

(FM = 0 only)

STORE: hq:=ha. The least significant 16 bits of the accumulator (A_L) are stored.

Condition Markers

N – *is set to the most significant bit of the halfword stored*
Z – *is set if the halfword stored is zero otherwise it is cleared*
CA – *not affected*
OF – *is set if the 16 bit halfword stored is not equal in value to the original 32 bit content of A.*

STW

0 5	6 15
1 0 1 1 1 1	ADDR

(FM = 0 only)

STORE WORD: Wq := a. The 32 bit accumulator (A_M and A_L) is stored.

Condition Markers

N – *is set to the sign of the 32 bit accumulator*
Z – *is set if the 32 bit accumulator is zero otherwise it is cleared*
CA – *not affected*
OF – *not affected*

AD

0 5	6 15
1 0 0 0 0 1	ADDR

(FM = 0 only)

ADD: a := a + hq. The halfword operand from store is sign extended to 32 bits before being added to the contents of the accumulator. Arithmetic is performed over 32 bits.

Condition Markers

N – *is set if the true result is negative*
Z – *is set if the result in the 32 bit accumulator is zero, otherwise it is cleared*
CA – *is set if there is a carry out of the most significant bit of the 32 bit accumulator otherwise it is cleared*
OF – *is set if arithmetic overflow occurs otherwise it remains unchanged.*

SB

0 5	6 15
1 0 0 0 1 0	ADDR

(FM = 0 only)

SUBTRACT: a := a - hq. The halfword operand from store is sign extended to 32 bits before being subtracted from the contents of the accumulator. Arithmetic is performed over 32 bits.

Condition Markers

N – *is set if the true result is negative*
Z – *is set if the result in the 32 bit accumulator is zero, otherwise it is cleared*
CA – *is set if there is a borrow out of the most significant bit of the 32 bit accumulator otherwise it is cleared.*
OF – *is set if arithmetic overflow occurs otherwise it remains unchanged.*

ADW

0 5	6 15
1 0 1 0 0 1	ADDR

(FM = 0 only)

ADD WORD: a := a + wq. The 32 bit operand from store is added to to contents of the 32 bit accumulator.

Condition Markers

N – *is set if the true result is negative*
Z – *is set if the result in the 32 bit accumulator is zero, otherwise it is cleared*
CA – *is set if there is a carry out of the most significant bit of the 32 bit accumulator otherwise it is cleared.*
OF – *is set if arithmetic overflow occurs otherwise it remains unchanged.*

SBW

0 5	6 15
1 0 1 0 1 0	ADDR

(FM = 0 only)

SUBTRACT WORD: a := a - wq. The 32 bit operand from store is subtracted from the contents of the 32 bit accumulator.

Condition Markers

N – *is set if the true result is negative*
Z – *is set if the result in the 32 bit accumulator is zero, otherwise it is cleared*
CA – *is set if there is a borrow out of the most significant bit of the 32 bit accumulator otherwise it is cleared*
OF – *is set if arithmetic overflow occurs otherwise it remains unchanged.*

CPB

0 5	6 15
0 1 0 0 1 1	ADDR

COMPARE BYTE: form ba - bq. The byte from store is extended to 32 bits by most significant zeros before being compared with the contents of the 32 bit accumulator. The contents of the accumulator are unaffected by this instruction.

Condition Markers

N – *is set if the 32 bit accumulator is less than the zero extended byte from store, otherwise it is cleared.*
Z – *is set if the 32 bit accumulator is equal to the zero extended byte from store, otherwise it is cleared.*
CA – *not affected*
OF – *not affected*

CP

0 5	6 15
1 0 0 0 1 1	ADDR

(FM = 0 only)

COMPARE: form a - hq. The halfword operand from store is sign extended to 32 bits before being compared with the contents of the 32 bit accumulator. The contents of the accumulator are unaffected by this instruction.

Condition Markers

N – *is set if the 32 bit accumulator is less than the sign extended halfword operand from store, otherwise it is cleared.*
Z – *is set if the 32 bit accumulator equals the sign extended operand from store, otherwise it is cleared.*
CA – *is set if there is a borrow out of the most significant bit of the function unit as a result of the comparison otherwise it is cleared.*
OF – *is set if arithmetic overflow occurs otherwise it remains unchanged.*

CPW

0 5	6 15
1 0 1 0 1 1	ADDR

(FM = 0 only)

COMPARE WORD: form a - wq. A 32 bit operand from store is compared with the contents of the 32 bit accumulator.

Condition Markers

N – *is set if the 32 bit accumulator is less than the 32 bit operand from store.*
Z – *is set if the 32 bit accumulator is equal to the 32 bit operand from store, otherwise it is cleared.*
CA – *is set if there is a borrow out of the most significant bit of the function unit as a result of the comparison otherwise it is cleared.*
OF – *is set if arithmetic overflow occurs otherwise it remains unchanged.*

N

0 5	6 15
1 0 0 1 0 0	ADDR

(FM = 0 only)

AND: a := a ∧ hq. The halfword operand from store is sign extended to 32 bits before being used to form the logical AND function with the contents of 32 bit accumulator. The result is placed in the accumulator.

Condition Markers

N – *is set to the sign of the 32 bit accumulator after the operation.*
Z – *is set if the 32 bit accumulator is zero after the operation, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

NW

0 5	6 15	
1 0 1 1 0 0	ADDR	(FM = 0 only)

AND WORD: a := a ∧ wq. The 32 bit operand from store is used to perform the logical AND function with the contents of the 32 bit accumulator. The result is placed in the accumulator.

Condition Markers

N – *is set to the sign of the 32 bit accumulator after the operation.*
Z – *is set if the 32 bit accumulator is zero after the operation, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

M

0 5	6 15	
1 0 0 1 0 1	ADDR	(FM = 0 only)

MULTIPLY: a := a * hq. The halfword operand from store (Multiplier) and the contents of the 32 bit accumulator (multiplicand) are multiplied together to form a 48 bit product. The result (in the accumulator) is the least significant 32 bits of the true product.

Condition Markers

N – *is set if the true product in A is negative.*
Z – *is set if the least significant 32 bits (i.e. the result in the accumulator) is zero, otherwise it is cleared.*
CA – *not affected.*
OF – *is set if significant bits are lost by truncating the product from 48 to 32 bits.*

MW

0 5	6 15	
1 0 1 1 0 1	ADDR	(FM = 0 only)

MULTIPLY WORD: da := a * wq. The 32 bit operand from store (Multiplier) and the contents of the 32 bit accumulator (multiplicand) are multiplied together to form a 64 bit product. The result is held in the 64 bit extended accumulator (BA) with the most significant bits in B and the least significant 32 bits in A.

Condition Markers

N – *is set if the true result in A is negative.*
Z – *is set if the result in BA is zero, otherwise it is cleared.*
CA – *not affected.*
OF – *cannot occur since the result in BA is the true 64 bit integer product.*

D

0 5	6 15	
1 0 0 1 1 0	ADDR	(FM = 0 only)

DIVIDE : a:= a ÷ hq b := remainder. The contents of the 32 bit accumulator (dividend) are divided by a halfword from store (divisor) and the result in the accumulator is the 32 bit integer quotient. The 32 bit remainder is held in B. The sign of the remainder is always equal to the sign of the dividend.

For example:—

DIVIDEND	DIVISOR	QUOTIENT	REMAINDER
+5	+2	+2	+1
-5	+2	-2	-1
+5	-2	-2	+1
-5	-2	+2	-1

Condition Markers

N – *is set if the quotient in A is negative.*
Z – *is set if the 32 bit result is zero, otherwise it is cleared.*
CA – *not affected*
OF – *is set if the divisor is zero or if the divisor is -1 and the dividend is* 2^{31}*.*
Under these circumstances A, B, N and Z are undefined.

DW

0 5	6 15	
1 0 1 1 1 0	ADDR	(FM = 0 only)

DIVIDE WORD: a := da ÷ wq b := remainder. The contents of the 64 bit extended accumulator BA (dividend) are divided by the 32 bit operand from store (divisor). The result, in the accumulator, is the least significant 32 bits of the quotient and the 32 bit remainder is held in B. As in DIVIDE the sign of the remainder is always equal to the sign of the dividend.

Condition Markers

N – *is set if the quotient in A is negative.*
Z – *is set if the 32 bit accumulator is zero otherwise it is cleared*
CA – *not affected*
OF – *is set if the quotient is out of range i.e. if more than 32 bits are required to hold the resulting quotient. An example is all numbers greater than* 2^{31} *divided by +1. If overflow occurs A, B, N and Z are undefined.*

7.2 OPERATIONS ON OPERANDS IN STORE

The following instructions are available:—

NBS	–	AND BYTE IN STORE
OBS	–	OR BYTE IN STORE
XBS	–	EXCLUSIVE OR BYTE IN STORE
DECS	–	DECREMENT STORE
INCS	–	INCREMENT STORE

NBS

0 5	6 15
0 1 0 1 0 0	ADDR

AND BYTE IN STORE: bq:= bq ∧ ba. A byte operand from store is used to perform the logical AND function with the least significant byte of the accumulator (i.e. bits 8–15 of A_L). The result is returned to store and the accumulator is unchanged.

Condition Markers

N – *is cleared by the instruction.*
Z – *is set if the result of the operation (that is returned to store) is zero, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

OBS

0 5	6 15
0 1 0 1 0 1	ADDR

OR BYTE IN STORE: bq := bq V ba. A byte operand from store is used to perform the logical OR function with the least significant byte of the accumulator (i.e. bits 8–15 of A_L). The result is returned to store and the accumulator is unchanged.

Condition Markers

N – *is cleared by the instruction.*
Z – *is set if the result of the operation (that is returned to store) is zero, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

XBS

0 5	6 15
0 1 0 1 1 0	ADDR

EXCLUSIVE OR BYTE IN STORE: bq := bq $\not\equiv$ ba. A byte operand from store is used to perform the logical EXCLUSIVE OR function with the least significant byte of the accumulator (i.e. bits 8–15 of A_L). The result is returned to store and the accumulator is unchanged.

Condition Markers

N – *is cleared by the instruction*
Z – *is set if the result of the operation (that is returned to store) is zero, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

DECS

0 5	6 15
0 1 0 0 1 0	ADDR

DECREMENT STORE: hq := hq – 1. The halfword operand from store is decremented by 1 and returned to the store. None of the registers in the central processor are affected.

Condition Markers

N – *is set to the true sign of the decremented operand*
Z – *is set if the decremented operand is zero, otherwise it is cleared.*
CA – *is set if the subtraction causes a borrow out of the most significant bit of the halfword operand.*
OF – *is set if the subtraction causes arithmetic overflow.*

INCS

0 5	6 15
0 1 1 1 1 1	ADDR

INCREMENT STORE: hq := hq + 1. The halfword operand from store is incremented by 1 and returned to the store. None of the registers in the central processor are affected.

Condition Markers

N – *is set to the true sign of the incremented operand.*
Z – *is set if the incremented operand is zero, otherwise it is cleared.*
CA – *is set if the addltion causes a carry out of the most significant bit of the halfword operand.*
OF – *is set if the addition causes arithmetic overflow.*

7.3 OPERATIONS ON THE X REGISTER

The following instructions are provided:–

LDX	–	LOAD X REGISTER
LBX	–	LOAD BYTE TO X REGISTER
STX	–	STORE X REGISTER
ADX	–	ADD X REGISTER
SBX	–	SUBTRACT X REGISTER
MX	–	MULTIPLY X REGISTER
DX	–	DIVIDE X REGISTER
NX	–	AND X REGISTER
CPX	–	COMPARE X REGISTER

LDX

0 5	6 15
1 1 0 0 0 0	ADDR

LOAD X REGISTER: x := hq. The 16 bit X register is loaded with a halfword operand from store.

Condition Markers

N	–	*is set to the sign of the 16 bit X register after the operation.*
Z	–	*is set if X is zero after the operation otherwise it is cleared.*
CA	–	*not affected.*
OF	–	*not affected.*

LBX

0 5	6 15
0 1 0 0 0 1	ADDR

LOAD BYTE TO X REGISTER: x := bq. A byte operand from store is loaded into the least significant byte of the X register. The operand byte is expanded to 16 bits by most significant zeros before being loaded into the X register.

Condition Markers

N	–	*is cleared by the instruction.*
Z	–	*is set if the X register is zero after the operation, otherwise it is cleared.*
CA	–	*not affected.*
OF	–	*not affected.*

STX

0 5	6 15
1 1 0 1 1 1	ADDR

STORE X REGISTER: hq := x. The 16 bit register is stored at a halfword address.

Condition Markers

N	–	*is set to the sign of the stored operand.*
Z	–	*is set if the stored operand is zero, otherwise it is cleared.*
CA	–	*not affected.*
OF	–	*not affected.*

ADX

0 5	6 15
1 1 0 0 0 1	ADDR

ADD TO X REGISTER: x := x + hq. The halfword operand from store is added to the contents of the 16 bit X register.

Condition Markers

N	–	*is set to the true sign of the X register after the operation.*
Z	–	*is set if the X register is zero after the operation otherwise it is cleared.*
CA	–	*is set if there is a carry out of the most significant bit of the X register, otherwise it is cleared.*
OF	–	*is set if arithmetic overflow occurs due to the operation on X.*

SBX

0 5	6 15
1 1 0 0 1 0	ADDR

SUBTRACT FROM X REGISTER: x := x - hq. The halfword operand from store is subtracted from the contents of the 16 bit X register.

Condition Markers

N – *is set to the true sign of the X register after the operation.*
Z – *is set if the X register is zero after the operation, otherwise it is cleared.*
CA – *is set if there is a borrow out of the most significant bit of the X register, otherwise it is cleared.*
OF – *is set if arithmetic overflow occurs due to the operation on X.*

CPX

0 5	6 15
1 1 0 0 1 1	ADDR

COMPARE X REGISTER: form x - hq. The halfword operand from store is compared with the contents of the 16 bit X register. The X register is unaffected by this instruction.

Condition Markers

N – *is set if the X register is less than the 16 bit operand from store*
Z – *is set if the 16 bit operand from store is equal to the X register, otherwise it is cleared.*
CA – *is set if there is a borrow out of the most significant bit of the function unit as a result of the comparison, otherwise it is cleared.*
OF – *is set if arithmetic overflow occurs as a result of the comparison.*

NX

0 5	6 15
1 1 0 1 0 0	ADDR

AND X REGISTER: x := x ∧ hq. The halfword operand from store is used to form the logical AND function with the contents of the X register. The result is replaced in the X register.

Condition Markers

N – *is set to the sign of the 16 bit X register after the operation.*
Z – *is set if X is zero after the operation otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

MX

0 … 5	6 … 15
1 1 0 1 0 1	ADDR

MULTIPLY X: x := x * hq. The halfword operand from store (multiplier) and the contents of the 16 bit X register (multiplicand) are multiplied together to form a 32 bit product. The result (in the X register) is the least significant 16 bits of the true product.

Condition Markers

N	–	*is set to the sign of the true product*
Z	–	*is set if the result in the X register is zero, otherwise it is cleared.*
CA	–	*not affected*
OF	–	*is set if significant bits are lost as a result of truncating the product from 32 to 16 bits.*

DX

0 … 5	6 … 15
1 1 0 1 1 0	ADDR

DIVIDE X REGISTER: x := x÷ hq. The contents of the 16 bit X register (Dividend) are divided by a halfword operand from store (divisor) and the result in X is the integer quotient. All remainders are discarded, the result being rounded towards zero.

Condition Markers

N	–	*is set to the sign of the quotient in X.*
Z	–	*is set if the quotient in X is zero otherwise it is cleared.*
CA	–	*not affected.*
OF	–	*is only set if the operand from store is zero or if the divident = -2^{15} and the divisor = -1. In this case N, Z and X are undefined.*

7.4 OPERATIONS ON Y AND Z REGISTERS

The following instructions are available:—

LDY	–	LOAD Y REGISTER
STY	–	STORE Y REGISTER
ADY	–	ADD Y REGISTER
SBY	–	SUBTRACT Y REGISTER
HAY	–	LOAD ADDRESS INTO Y REGISTER
LDZ	–	LOAD Z REGISTER
STZ	–	STORE Z REGISTER
ADZ	–	ADD Z REGISTER
SBZ	–	SUBTRACT Z REGISTER
HAZ	–	LOAD ADDRESS INTO Z REGISTER

LDY

0 5	6 15
1 1 1 0 0 0	ADDR

LOAD Y REGISTER: y := hq. The 16 bit register Y is loaded with a halfword operand from store.

Condition Markers

Not affected.

STY

0 5	6 15
1 1 1 0 1 1	ADDR

STORE Y REGISTER: hq := y. The content of the 16 bit Y register is stored at a halfword address.

Condition Markers

Not affected.

ADY

0 5	6 15
1 1 1 0 0 1	ADDR

ADD Y REGISTER: y := y + hq. The halfword operand from store is added to the contents of the 16 bit Y register.

Condition Markers

Not affected

SBY

0 5	6 15
1 1 1 0 1 0	ADDR

SUBTRACT Y REGISTER: y := y - hq. The halfword operand from store is subtracted from the contents of the 16 bit Y register.

Condition Markers

Not affected

HAY

0					5	6 15
0	1	1	0	0	0	ADDR

LOAD HALFWORD ADDRESS INTO Y REGISTER: y := Q. The halfword operand address specified by the instruction is loaded into Y.

Condition Markers

Not affected

LDZ

0					5	6 15
1	1	1	1	0	0	ADDR

LOAD Z REGISTER: z := hq. The 16 bit register Z is loaded with a halfword operand from store.

Condition Markers

Not affected

STZ

0					5	6 15
1	1	1	1	1	1	ADDR

STORE Z REGISTER: hq := z. The content of the 16 bit Z register is stored at a halfword address.

Condition Markers

Not affected

ADZ

0					5	6 15
1	1	1	1	0	1	ADDR

ADD Z REGISTER: z := z + hq. The halfword operand from store is added to the contents of the 16 bit Z register.

Condition Markers

Not affected

SBZ

0 5	6 15
1 1 1 1 1 0	ADDR

SUBTRACT Z REGISTER: z := z - hq. The halfword operand from store is subtracted from the contents of the 16 bit Z register.

Condition Markers

Not affected

HAZ

0 5	6 15
0 1 1 1 0 0	ADDR

LOAD HALFWORD ADDRESS INTO Z REGISTER: z := Q. The halfword operand address specified by the instruction is loaded into Z.

Condition Markers

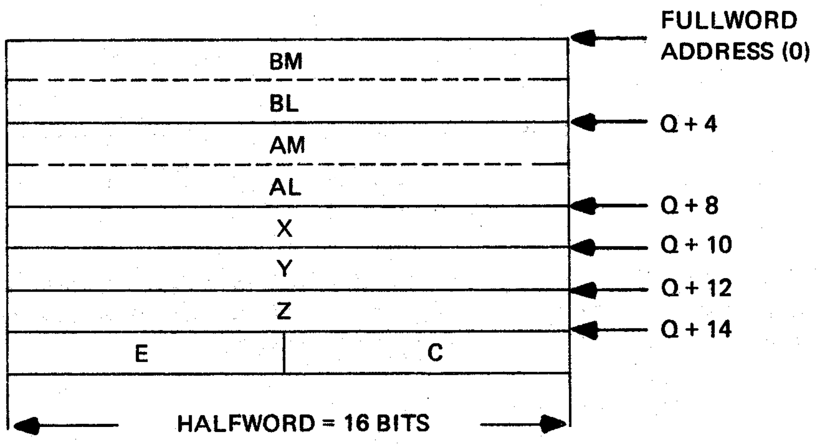
Not affected

7.5 MULTIPLE LOAD AND STORE INSTRUCTIONS

Two instructions are available:–

STM – STORE MULTIPLE
LDM – LOAD MULTIPLE

9 registers take part in each of these instructions and the store layout is as below.



STM

0 5	6 15
0 1 1 0 1 0	ADDR

STORE MULTIPLE: The displacement field of the instruction is scaled to provide a fullword address. The address formed by instructions in formats A1–A5 will be truncated to a fullword address. Register BM is stored at this fullword location followed by the registers shown above in successive halfword locations.

All registers are unaffected by the STORE MULTIPLE instruction.

Condition Markers

Not affected.

LDM

0 5	6 15
0 1 1 0 0 1	ADDR

LOAD MULTIPLE: The displacement field of the instruction is scaled to provide a fullword address. The address formed by instructions in formats A1–A5 will be truncated to a fullword address. Register BM is loaded from this fullword location followed by the registers shown above in successive halfword locations.

Condition Markers

The Condition Markers,including FM, are loaded from the Q + 15 store location during LDM.

7.6 INDIRECT BRANCHES

Two instructions are available:–

BI	–	BRANCH INDIRECT
BLI	–	BRANCH AND LINK INDIRECT

BI

0 5	6 15
0 1 1 1 0 1	ADDR

BRANCH INDIRECT: s := hq. A halfword address is formed as specified by the instruction. This is an operand address formed in the same way as for any format A1–A5 instruction. The contents of this location is the branch destination, and is loaded into the sequence control register (S).

Condition Markers

Not affected.

BLI

0 5	6 15
0 1 1 1 1 0	ADDR

BRANCH AND LINK INDIRECT: z := s s := hq. A halfword address is formed exactly as in BI above. The contents of the sequence control register S is transferred to Z before S is loaded with the branch destination address.

This instruction is used for subroutine entry with the return link stored in Z.

Condition Markers

Not affected.

8 INSTRUCTIONS AVAILABLE IN FORMAT B

Two unconditional branch instructions only are available in this format:—

B – BRANCH
BL – BRANCH AND LINK

B

0 5	6 15
0 0 0 0 0 1	D

(Signed integer -10 bits)

Destination ADDress=S±2D
511 Halfwords Forwards
512 Halfwords Backwards

BRANCH: s := Q. The sequence control register is incremented or decremented according to the displacement field specified by the instruction. As explained in section 4 (b) the instruction provides unconditional branches relative to the next instruction in sequence. The range of the branch is limited to 511 halfwords forwards and 512 halfwords backwords by the 10 bits of instruction displacement field.

Condition Markers

Not affected.

BL

0 5	6 15
0 0 0 0 1 0	D

BRANCH AND LINK: z := s. This instruction performs the same function as B above, and also the original value of the sequence control register is placed in Z. This provides for subroutine entry and return.

Condition Markers

Not affected.

9 INSTRUCTIONS AVAILABLE IN FORMAT L

These instructions are called literal instructions since where an operand is required it is specified exactly by the instruction displacement field. Some instructions depend on their displacement fields to further define their function.

9.1 OPERATIONS ON THE 32 BIT ACCUMULATOR

The following instructions are available:–

LDL	–	LOAD LITERAL
ADL	–	ADD LITERAL
SBL	–	SUBTRACT LITERAL
ML	–	MULTIPLY LITERAL
DL	–	DIVIDE LITERAL
NL	–	AND LITERAL
CPL	–	COMPARE LITERAL

LDL

0 1	2 7	8 15
0 0	1 0 0 0 0 0	D

LOAD LITERAL: a := D. The operand formed by the 8-bit displacement field is loaded into the least significant byte of the accumulator. The most significant 3 bytes of the accumulator are cleared. As described in section 4.2 it is thus possible to load numbers in the range 0–255 into the 32 bit accumulator.

Condition Markers

N – *is cleared by the operation.*
Z – *is set if the result in the 32 bit accumulator is zero, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

ADL

0 1	2 7	8 15
0 0	1 0 0 0 0 1	D

ADD LITERAL: a := a + D. The operand specified by the 8-bit displacement field is extended to 32 bits by most significant zeros before the operation. This operand is then added to the contents of the 32 bit accumulator.

Condition Markers

N – *is set to the true sign of the result.*
Z – *is set if the result in the 32 bit accumulator is zero, otherwise it is cleared.*
CA – *is set if there is a carry out of the most significant bit of the 32 bit accumulator, otherwise it is cleared.*
OF – *is set if arithmetic overflow occurs, otherwise it remains unchanged.*

SBL

0 1	2 7	8 15
0 0	1 0 0 0 1 0	D

SUBTRACT LITERAL: a := a - D. The operand specified by the 8-bit displacement field is extended to 32 bits by most significant zeros before the operation. This operand is then subtracted from the contents of the 32 bit accumulator.

Condition Markers

N – *is set to the true sign of the result.*
Z – *is set if the result in the 32 bit accumulator is zero, otherwise it is cleared.*
CA – *is set if there is a borrow out of the most significant bit of the 32 bit accumulator, otherwise it is cleared.*
OF – *is set if arithmetic overflow occurs, otherwise it remains unchanged.*

CPL

0 1	2 7	8 15
0 0	1 0 0 0 1 1	D

COMPARE LITERAL: form a- D. The operand specified by the instruction displacement field is extended to 32 bits by most significant zeros. This operand is then compared with the contents of the 32 bit accumulator. The accumulator is unaffected by the instruction.

Condition Markers

N – *is set if the 32 bit accumulator is less than the 32 bit literal operand, otherwise it is cleared.*
Z – *is set if the 32 bit accumulator equals the 32 bit literal operand, otherwise it is cleared.*
CA – *is set if there is a borrow out of the most significant bit of the function unit as a result of the comparison, otherwise it is cleared.*
OF – *is set if arithmetic overflow occurs, otherwise it remains unchanged.*

NL

0 1	2 7	8 15
0 0	1 0 0 1 0 0	D

AND LITERAL: a := a ∧ D. The operand specified by the displacement field of the instruction is extended to 32 bits by most significant zeros. This 32 bit operand is used to perform the logical AND function with the contents of the 32 bit accumulator. The result is placed in the accumulator.

Condition Markers

N – *is cleared by the instruction.*
Z – *is set if the result in the accumulator is zero, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

ML

0 1	2 7	8 15
0 0	1 0 0 1 0 1	D

MULTIPLY LITERAL: a := a * D. The 8-bit operand specified by the displacement field of the instruction and the contents of the 32 bit accumulator are multiplied together to form a 40 bit product. The result, in the accumulator, is the least significant 32 bits of the true product.

Condition Markers

N	–	*is set to the sign of the true 40 bit result.*
Z	–	*is set if the result in the 32 bit accumulator is zero, otherwise it is cleared.*
CA	–	*not affected.*
OF	–	*overflow is set if significant bits are lost in the truncation of the product.*

DL

0 1	2 7	8 15
0 0	1 0 0 1 1 0	D

DIVIDE LITERAL: a := a ÷ D b := remainder. The contents of the 32 bit accumulator (dividend) are divided by the 8-bit operand specified by the instruction displacement field. This 8-bit operand is extended to 16 bits before the operation by most significant zeros. The result, in the accumulator, is the 32 bit integer quotient. The result in B is the 16 bit remainder, sign extended to 32 bits. The sign of the remainder is always equal to the sign of the dividend (see section 7.1).

Condition Markers

N	–	*is set to the sign of the 32 bit quotient in the accumulator.*
Z	–	*is set if the quotient is zero, otherwise it is cleared.*
CA	–	*not affected.*
OF	–	*overflow is set if the divisor is zero. The content of A, B, N and Z are then undefined.*

9.2 LITERAL OPERATIONS ON THE X REGISTER

The following instructions are available:–

LDXL	–	LOAD X LITERAL
ADXL	–	ADD X LITERAL
SBXL	–	SUBTRACT X LITERAL
MXL	–	MULTIPLY X LITERAL
DXL	–	DIVIDE X LITERAL
NXL	–	AND X LITERAL
CPXL	–	COMPARE X LITERAL

LDXL

0 1	2 7	8 15
0 0	1 1 0 0 0 0	D

LOAD X LITERAL: x := D. The operand specified by the 8-bit displacement field is extended to halfword length (16 bits) by most significant zeros. This 16 bit operand is loaded into the 16 bit X register.

Condition Markers

N – *is cleared by the instruction.*
Z – *is set if the X register is made zero by the operation, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

ADXL

0 1	2 7	8 15
0 0	1 1 0 0 0 1	D

ADD X LITERAL: x := x + D. The operand specified by the 8-bit displacement field is extended to 16 bits by most significant zeros. This 16 bit operand is then added to the contents of the 16 bit X register.

Condition Markers

N – *is set to the true sign of the result.*
Z – *is set if the result in the X register is zero, otherwise it is cleared.*
CA – *is set if there is a carry out of the most significant bit of the X register, otherwise it is cleared.*
OF – *is set if arithmetic overflow occurs due to the operation in X.*

SBXL

0 1	2 7	8 15
0 0	1 1 0 0 1 0	D

SUBTRACT X LITERAL: x := x - D. The operand specified by the 8 bit displacement field is extended to 16bits by most significant zeros. This 16 bit operand is then subtracted from the contents of the 16 bit X register.

Condition Markers

N – *is set to the true sign of the result.*
Z – *is set if the result in the X register is zero, otherwise it is cleared.*
CA – *is set if there is a borrow out of the most significant bit of the X register, otherwise it is cleared.*
OF – *is set if arithmetic overflow occurs due to the operation in X.*

MXL

0 1	2 7	8 15
0 0	1 1 0 1 0 1	D

MULTIPLY X LITERAL: x := x * D. The operand specified by the displacement field of the instruction and the contents of the 16 bit X register are multiplied together to form a 24 bit product. The result, in X, is the least significant 16 bits of this 24 bit product.

Condition Marker

N – is set to the sign of the true 24 bit product.
Z – is set if the result in the X register is zero, otherwise it is cleared.
CA – not affected.
OF – is set if significant bits are lost in the truncation from 24 to 16 bits.

DXL

0	1	2					7	8	15
0	0	1	1	0	1	1	0	D	

DIVIDE X LITERAL: x := x ÷ D. The contents of the 16 bit X register (dividend) are divided by the 8 bit operand (divisor) specified by the instruction displacement field. This 8 bit operand is extended to 16 bits before the operation by most significant zeros. The result, in X, is the true integer quotient. Remainders are discarded, the result being rounded towards zero.

Condition Markers

N – is set to the sign of the 16 bit quotient in the X register.
Z – is set if the result in the X register is zero, otherwise it is cleared.
CA – not affected.
OF – is set if the divisor is zero. The contents of N, Z and X are then undefined.

NXL

0	1	2					7	8	15
0	0	1	1	0	1	0	0	D	

AND X LITERAL: x := x ∧ D. The operand specified by the 8 bit displacement field is extended to 16 bits by most significent zeros. This 16 bit operand is used to form the logical AND function with the contents of the 16 bit X register. The result is placed in the X register.

Condition Markers

N – is cleared by the operation.
Z – is set if the result in the X register is zero, otherwise it is cleared.
CA – not affected.
OF – not affected.

CPXL

0	1	2					7	8	15
0	0	1	1	0	0	1	1	D	

COMPARE X LITERAL: form x - D. The operand specified by the 8 bit displacement field is extended to 16 bits by most significant zeros. This 16 bit operand is then compared with the 16 bit content of the X register. The X register is unaffected by the operation.

Condition Markers

N – is set if the 16 bit X register is less than the 16 bit literal operand, otherwise it is cleared.
Z – is set if the 16 bit X register equals the 16 bit literal operand, otherwise it is cleared.
CA – is set if there is a borrow out of the most significant bit of the function unit as a result of the comparison, otherwise it is cleared.
OF – is set if arithmatic overflow occurs.

9.3 LITERAL OPERATIONS ON THE Y AND Z REGISTERS

The following instructions are available:—

LDYL	–	LOAD Y LITERAL
ADYL	–	ADD Y LITERAL
SBYL	–	SUBTRACT Y LITERAL
CPYL	–	COMPARE Y LITERAL
LDZL	–	LOAD Z LITERAL
ADZL	–	ADD Z LITERAL
SBZL	–	SUBTRACT Z LITERAL
CPZL	–	COMPARE Z LITERAL

N.B. Before all the following operations the 8 bit displacement field of the instruction is extend to 16 bits by most significant zeros. It is then used as a 16 bit literal operand.

LDYL

0	1	2					7	8	15
0	0	1	1	1	0	0	0	D	

LOAD Y LITERAL: y := D. The literal operand is loaded into the 16 bit Y register.

Condition Markers

Not affected.

ADYL

0	1	2					7	8	15
0	0	1	1	1	0	0	1	D	

ADD Y LITERAL: y := y + D. The literal operand is added to the contents of the 16 Y register.

Condition Markers

Not affected.

SBYL

0	1	2					7	8	15
0	0	1	1	1	0	1	0	D	

SUBTRACT Y LITERAL: y := y - D. The literal operand is subtracted from the contents of the 16 bit Y register.

Condition Markers

Not affected.

CPYL

0	1	2					7	8	15
0	0	0	1	1	0	1	1	D	

COMPARE Y LITERAL: form y - D. The literal operand is compared with the contents of the Y register. Y is unaffected by this operation.

Condition Markers

N	–	*is set if Y is less than the literal operand.*
Z	–	*is set if the result of the operation is zero, otherwise it is cleared.*
CA	–	*is set if there is a borrow out of the most significant bit of the function unit as a result of the comparison, otherwise it is cleared.*
OF	–	*is set if arithmetic overflow occurs.*

LDZL

0	1	2					7	8	15
0	0	1	1	1	1	1	1	D	

LOAD Z LITERAL: z := D. The literal operand is loaded into the 16 bit Z register.

Condition Markers

Not affected.

ADZL

0	1	2					7	8	15
0	0	1	1	1	1	0	1	D	

ADD Z LITERAL: z := z + D. The literal operand is added to the contents of the 16 bit Z register.

Condition Markers

Not affected.

SBZL

0	1	2					7	8	15
0	0	1	1	1	1	1	0	D	

SUBTRACT Z LITERAL: z := z - D. The literal operand is subtracted from the contents of the 16 bit Z register.

Condition Markers

Not affected.

CPZL

0	1	2					7	8	15
0	0	0	1	1	1	1	1	D	

COMPARE Z LITERAL: form z - D. The literal operand is compared with the contents of the Z register. Z is unaffected by this operation.

Condition Markers

N – *is set if Z is less than the literal operand.*
Z – *is set if the result of the operation is zero, otherwise it is cleared.*
CA – *is set if there is a borrow out of the most significant bit of the function unit as a result of the comparison, otherwise it is cleared.*
OF – *is set if arithmetic overflow occurs.*

9.4 CONDITION BRANCH INSTRUCTIONS IN FORMAT L

The following instructions are available:–

BN	–	BRANCH IF NEGATIVE
BNN	–	BRANCH IF NON NEGATIVE
BZ	–	BRANCH IF ZERO
BNZ	–	BRANCH IF NON ZERO
BP	–	BRANCH POSITIVE
BNP	–	BRANCH IF NON POSITIVE
BOF	–	BRANCH IF OVERFLOW
BNCA	–	BRANCH IF NO CARRY
BPAR	–	BRANCH ON ODD PARITY

For these instructions the branch destination is formed by scaling the literal displacement (D) for halfword (i.e. left shifting it one place), sign extending it to 16 bits and then adding it to the current contents of the sequence control register (S).

Branches of 127 instructions ahead or 128 instructions behind the next instruction in sequence may be performed (see section 4.3).

If a branch is taken then

S = S + 2D where D = +127 . . . . -128

otherwise for a branch not taken the next instruction in sequence is obeyed.

BN

0	1	2					7	8	15
0	0	1	0	1	0	0	0	D 8 bits	

D= +127 . . . -128

BRANCH IF NEGATIVE: if N then s := Q. If the negative condition marker is true the branch is taken.

Condition Markers

Not affected.

BNN

0 1	2 7	8 15
0 0	1 0 1 0 0 1	D

BRANCH IF NON NEGATIVE: if $\overline{N}$ then s := Q. If the negative condition marker is false the branch is taken.

Condition Markers

Not affected.

BZ

0 1	2 7	8 15
0 0	1 0 1 0 1 0	D

BRANCH IF ZERO: if Z then s := Q. If the zero condition marker is true the branch is taken.

Condition Markers

Not affected.

BNZ

0 1	2 7	8 15
0 0	1 0 1 0 1 1	D

BRANCH IF NON ZERO: if $\overline{Z}$ then s := Q. If the zero condition marker is false the branch is taken.

Condition Markers

Not affected.

BP

0 1	2 7	8 15
0 0	1 0 1 1 0 0	D

BRANCH IF POSITIVE: if $\overline{N} \wedge \overline{Z}$ then s := Q. If the zero condition marker AND the negative marker are false the branch is taken.

Condition Markers

Not affected.

BNP

0 1	2 7	8 15
0 0	1 0 1 1 0 1	D

BRANCH IF NON POSITIVE: if N ∨ Z then s := Q. If the zero condition marker OR the negative marker are true then the branch is taken.

Condition Markers

Not affected.

BOF

0 1	2 7	8 15
0 0	1 0 1 1 1 0	D

BRANCH IF OVERFLOW: if OF then ≪OF := 0 s := Q≫. If the overflow condition marker (OF) is true then the branch is taken.

Condition Markers

N	–	*not affected.*
Z	–	*not affected.*
CA	–	*not affected.*
OF	–	*resets to zero.*

BNCA

0 1	2 7	8 15
0 0	1 0 1 1 1 1	D

BRANCH IF NO CARRY: if $\overline{CA}$ then s := Q. If the carry condition marker (CA) is false then the branch is taken.

Condition Markers

Not affected.

BPAR

0 1	2 7	8 15
0 0	0 1 1 0 0 0	D

BRANCH ON ODD PARITY: The least significant byte of the accumulator is examined and if it has odd parity the branch is taken.The parity of the least significant byte of A is defined as odd, if the number of bits which are set to 1 are odd.

Condition Markers

Not affected.

9.5 SHIFT INSTRUCTIONS

The following instructions are available:—

SHIFT LITERAL
SHIFT INDEXED

SHIFT LITERAL

0 1	2 7	8 15
0 0	0 1 1 0 0 1	D

SHIFT LITERAL: D Defines Shift. The type, direction and number of places to be shifted are further defined by the displacement field of the instruction. The 8 bit literal operand directly controls the shift operation.

SHIFT INDEXED

0 1	2 7	8 15
0 0	1 1 1 0 1 1	D

SHIFT INDEXED:D + x Defines Shift. The type, direction and number of places to be shifted are further defined by the displacement field of the instruction added to the contents of the X register. The least significant byte of the result controls the shift operation.

The 8 bit field controlling the shift instruction is structured as follows:—

8 10	11 15
M	N

The 3 bit field M defines eight possible types of shift operation, and the 5 bit field N defines the number of places to be shifted.

In the case of the indexed shifts (SHX instruction) both the number and type of shift may be modified by the value in the X register. An effective subtraction of X may be achieved if the value of X is negative since the indexing involves a full 16 bit addition.

The following types of shift are available:–

M	MNEMONIC	NAME
000	SBAR,SBRX	SHIFT EXTENDED ACCUMULATOR (B AND A) RIGHT ARITHMETICAL
001	SBAL,SBLX	SHIFT EXTENDED ACCUMULATOR LEFT ARITHMETICAL
010	SR, SRX	SHIFT 32 BIT ACCUMULATOR (A) RIGHT ARITHMETICAL
011	SL, SLX	SHIFT 32 BIT ACCUMULATOR (A) LEFT ARITHMETICAL
100	SRL, ,SRLX	SHIFT 32 BIT ACCUMULATOR (A) RIGHT LOGICAL
101	SLC, SLCX	SHIFT 32 BIT ACCUMULATOR (A) LEFT CIRCULAR
110	SXR, SXRX	SHIFT X REGISTER RIGHT ARITHMETICAL
111	SXL, SXLX	SHIFT X REGISTER LEFT

EVEN = RIGHT SHIFT
ODD = LEFT SHIFT

SBAR,SBRX

0 7	8 10	11 15
SHL/SHX	0 0 0	N

SHIFT BA RIGHT ARITHMETICAL: The 64 bit accumulator is shifted arithmetically right a number of places defined by the N field. N may have the value 1–31 and the number of places shifted will be 32–N. If the value of N=0 is used the effect will be to set the condition markers N and Z as below leaving the accumulator unchanged. The most significant bit of the extended accumulator (sign bit) is replicated as the shift is performed.

Condition Markers

N	–	*is set to the sign of the 64 bit extended accumulator after the shift*
Z	–	*is set if as a result of the shift the 64 bit accumulator becomes zero, otherwise it is cleared.*
CA	–	*not affected.*
OF	–	*not affected.*

SBAL,SBLX

0 7	8 10	11 15
SHL/SHX	0 0 1	N

SHIFT BA LEFT ARITHMETICAL: The 64 bit accumulator is shifted left a number of places defined by the N field. N may have the value 0–31 and the number of placed shifted will equal N. Zeros are input at the least significant bit of the extended accumulator as it is shifted. If a value of N=0 is used the effect will be to set the condition markers N and Z as below leaving all other registers unchanged.

Condition Markers

N	–	*is set to the sign of the 64 bit extended accumulator after the shift.*
Z	–	*is set if as a result of the shift the 64 bit accumulator becomes zero, otherwise it is cleared.*
CA	–	*not affected.*
OF	–	*is set if in the course of the shift the sign of the extended accumulator changes.*

SR,SRX

0 — 7	8 — 10	11 — 15
SHL/SHX	0 1 0	N

SHIFT A RIGHT ARITHMETICAL: The 32 bit accumulator is shifted right arithmetically a number of places defined by the N field. N may have the value 1–31 and the number of places shifted will be 32–N. If a value of N=0 is used the effect is to set condition markers N and Z as below leaving the accumulator unchanged. The most significant bit of the accumulator is replicated as the shift is performed.

Condition Markers

N – *is set to the sign of the 32 bit accumulator after the shift.*
Z – *is set if as a result of the shift the 32 bit accumulator is zero, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

SL,SLX

0 — 7	8 — 10	11 — 15
SHL/SHX	0 1 1	N

SHIFT A LEFT ARITHMETICAL: The 32 bit accumulator is shifted left a number of places defined by the N field. N may have the value 0–31 and the number of places shifted will equal N. Zeros are input at the least significant bit of the accumulator as it is shifted.

If the value N=0 is used the effect will be to set the condition markers as below leaving the accumulator unchanged.

Condition Markers

N – *is set to the sign of the 32 bit accumulator after the shift.*
Z – *is set if as a result of the shift the 32 bit accumulator becomes zero, otherwise it is cleared.*
CA – *not affected.*
OF – *is set if in the course of the shift the sign of the extended accumulator changes.*

SRL,SRLX

0 — 7	8 — 10	11 — 15
SHL/SHX	1 0 0	N

SHIFT RIGHT LOGICAL: The 32 bit accumulator is shifted logically a number of places defined by the N field. N may have the value 1–31 and the number of places shifted will be 32–N. If the value N=0 is used the effect is to set the condition markers as below leaving all other registers unchanged. Zeros are input at the most significant bit of the 32 bit accumulator as it is shifted.

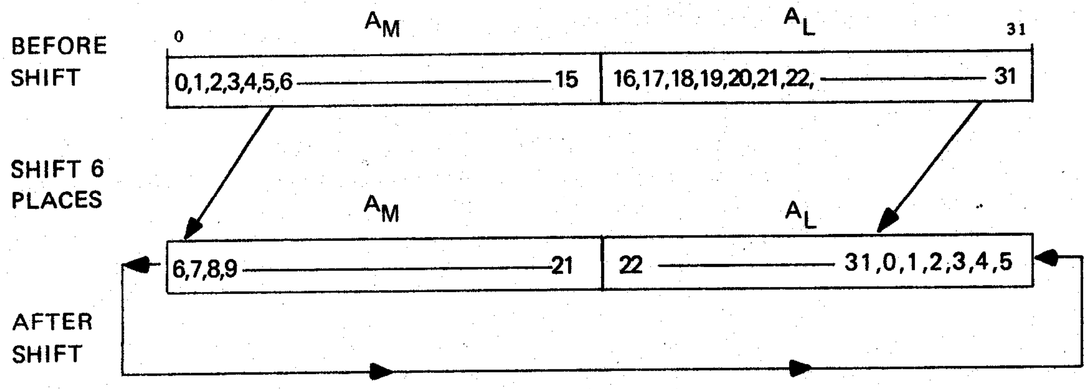
Condition Markers

N – *is set to the sign of the 32 bit accumulator after the shift.*
Z – *is set if the 32 bit accumulator becomes zero as a result of the shift. otherwise it is cleared.*
OF – *not affected.*
CA – *not affected.*

SLC,SLCX

0 7	8 10	11 15
SHL/SHX	1 0 1	N

SHIFT A LEFT CIRCULAR: The 32 bit accumulator is shifted left circular a number of places defined by the N field. N may have the value 0–31 and the number of places shifted will equal N. Bits shifted out from the most significant bit of the 32 bit accumulator are input at the least significant bit as the shift proceeds as shown



If the value of N=0 is used the effect is to set the condition markers N and Z as below leaving all other registers unchanged.

Condition Markers

N – *is set to the sign of the 32 bit accumulator after the shift (bit 6 in the example).*
Z – *is set if after the shift the accumulator is zero.*
OF – *not affected.*
CA – *not affected.*

SXR,SXRX

0 7	8 10	11 15
SHL/SHX	1 1 0	N

SHIFT X RIGHT ARITHMETIC (16 ≤ N ≤ 31). The 16 bit X register is shifted right arithmetically a number of places defined by the N field. N may have the value 16–31 and the number of places shifted will equal 32–N. Use of a value of N in the range 0–15 will lead to an undefined instruction trap. In this case all registers are unchanged. The most significant bit of the X register is replicated as the shift is performed.

Condition Markers

N – *is set to the sign of the X register after the shift.*
Z – *is set if as a result of the shift the X register becomes zero, otherwise it remains unchanged.*
CA – *not affected.*
OF – *not affected.*

SXL,SXLX

0 7	8 10	11 15
SHL/SHX	1 1 1	N

SHIFT X LEFT ARITHMETIC ($0 \leqslant N \leqslant 15$): The 16 bit X register is shifted left a number of places defined by the N field. N may have the value 0–15 and the number of places shifted will be equal to N. Use of a value of N in the range 16–31 will lead to undefined instruction trap. In this case all registers are unchanged. If a value of N=0 is used the effect will be to set condition markers as below leaving all other registers unchanged.

Condition Markers

N – is set to the sign of the X register after the shift.
Z – is set if as a result of the shift the X register becomes zero, otherwise it remains unchanged.
CA – not affected.
OF – not affected.

9.6 BIT MANIPULATION INSTRUCTIONS

The following instruction types are available:

BITL – BIT OPERATION LITERAL
BITX – BIT OPERATION INDEXED

These instructions perform operations on a selected bit of the accumulator. The least significant 16 bits (A_L) of the 32 bit accumulator only are affected by these instructions.

BITL

0 1	2 7	8 15
0 0	0 1 0 1 1 0	D

BIT OPERATION LITERAL: D defines OP. The operation to be performed and the bit selected are defined by the displacement field of the instruction. The 8 bit literal operand specifies the operation to be performed.

BITX

0 1	2 7	8 15
0 0	0 1 0 1 0 1	D

BIT OPERATION INDEXED: D + x defines OP. The operation to be performed and the bit selected are defined by the displacement field of the instruction added to the contents of the X register. The least significant byte of the result controls the operation. The operation type and the selected bit number may be modified by the value in the X register. If the OP field is modified, undefined operation may result. The indexing with X should therefore only be used to modify the bit number N.

The 8 bit field controlling the bit manipulation operation is structured as follows:–

8 11	12 15
OP	N

The 4 bit OP field defines the operation to be performed.

The 4 bit N field defines the bit of the accumulator (A_L) that is selected for the operation. The bits of A_L are numbered 0 at the most significant end to 15 at the least significant end.

The following bit manipulation instructions are specified:–

OP	MNEMONIC	NAME
0	TSTB,TSTX	TEST BIT
9	TGLB,TGLX	TOGGLE BIT
10	PLCB,PLCX	PLACE BIT
11	SETB,SETX	SET BIT
14	CLRB,CLRX	CLEAR BIT

TSTB,TSTX

0 7	8 11	12 15
BITL/BITX	0 0 0 0	N

TEST BIT: The bit of the accumulator (A_L) specified by the N field is tested.

Condition Markers

N – *not affected.*
Z – *is set if the bit tested is zero, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

TGLB,TGLX

0 7	8 11	12 15
BITL/BITX	1 0 0 1	N

TOGGLE BIT: The bit of the accumulator (A_L) specified by the N field is toggled, i.e. if the initial value was 0 the new value is 1 and if the initial value was 1 the new value is 0.

Condition Markers

N – *not affected.*
Z – *is set if the least significant 16 bits of the accumulator (A_L) become zero as a result of the operation otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

PLCB,PLCX

0 7	8 11	12 15
BITL/BITX	1 0 1 0	N

PLACE BIT: The accumulator (A_L) is loaded with a single bit specified by the N field. The specified bit is set to a 1 whilst the remaining bits are set to zero.

e.g.

	0 — 7	8 — 11	12 — 15
INSTRUCTION CODE	BIT L	OP	N
	0 0 0 1 0 1 1 0	1 0 1 0	1 1 0 0

N=12

	A_M	A_L
RESULT IN ACCUMULATOR	UNCHANGED	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0

Condition Markers

N – *not affected.*
Z – *will be cleared by this instruction since the result can never be zero.*
CA – *not affected.*
OF – *not affected.*

SETB,SETX

0 — 7	8 — 11	12 — 15
BITL/BITX	1 0 1 1	N

SET BIT: The bit of the accumulator (A_L) specified by the N field is set to a 1 whilst the remaining bits are unchanged.

Condition Markers

N – *not affected.*
Z – *is cleared as a result of the operation since the result in A_L can never be zero.*
CA – *not affected.*
OF – *not affected.*

CLRB,CLRX

0 — 7	8 — 11	12 — 15
BITL/BITX	1 1 1 0	N

CLEAR BIT: The bit of the accumulator A_L specified by the N field is set to 0 whilst the remaining bits are unchanged.

Condition Markers

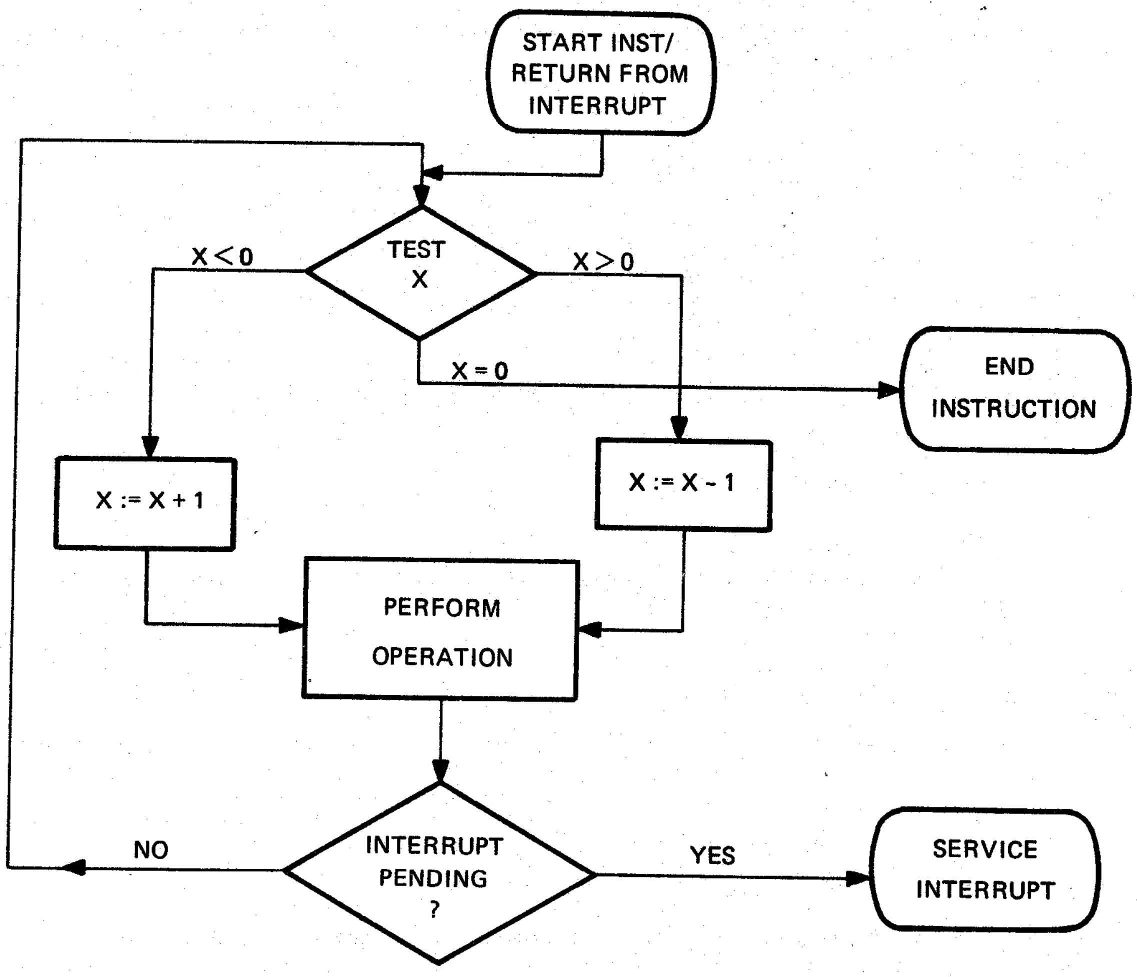
N – *not affected.*
Z – *is set if as a result of the operation the least significant 16 bits of the accumulator become zero, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

9.7 STRING MANIPULATION INSTRUCTIONS

These instructions operate on strings of bytes or halfwords in store. One string of bytes/halfwords has it position defined by the Y register and its length defined by the X register. The second string is defined by Z and its length is either defined by X or is fixed at 256 bytes depending on the instruction.

In general X is counted towards zero as each byte/halfword is operated on. When X = 0 the instruction terminates. At the end of each operation a test is made to determine the presence of any interrupt that may require attention. When the instruction is restarted (possibly after servicing an interrupt) the next operation in sequence is performed until X = 0.

A general flow chart for string manipulation instruction is shown below.



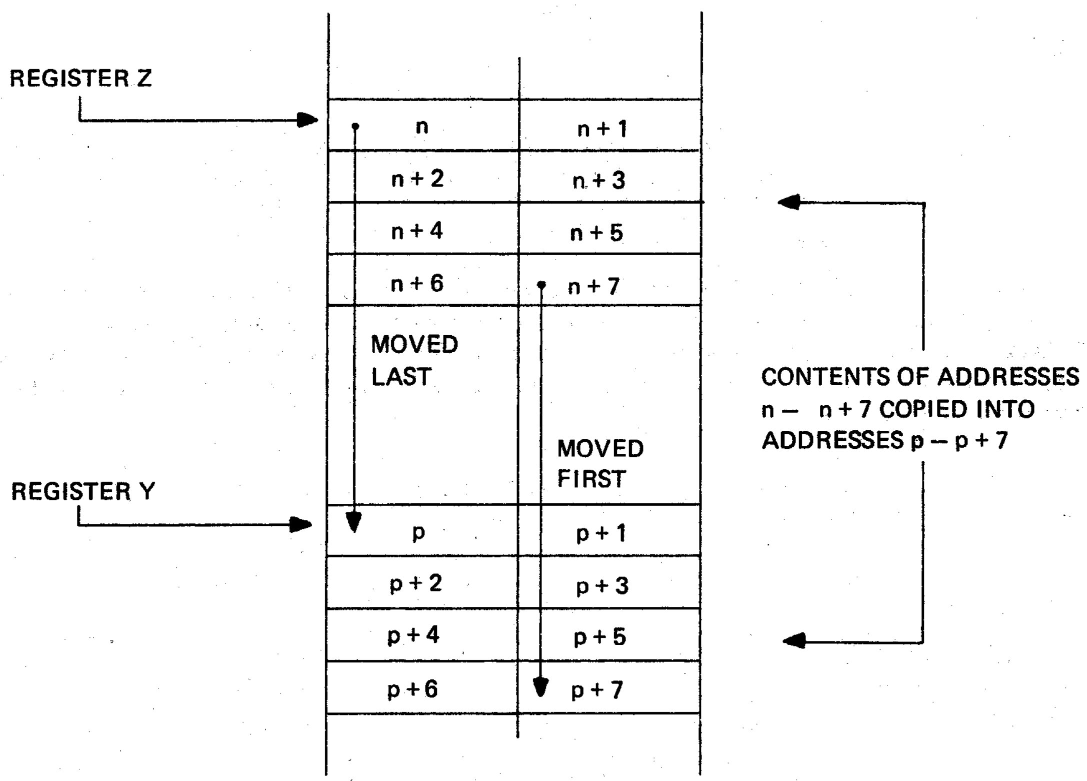
MBS

0	1	2					7	8							15
0	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0

b (y+x) :=b (z+x)

MOVE BYTE STRING: The string of bytes defined by register Z is copied into the string defined by register Y. The X register specifies the number of bytes to be moved. If X is positive Y and Z must contain the address of the first byte in each string and the first byte moved is the last byte in each string. If X is initially negative Y and Z must contain the address of the last byte in each string and the first byte moved is the first byte in each string. Thus the final position of the string may in either case overlap its original position.

If for example X = +8
Y = address p
Z = address n



Initially when X = 8 a byte from address n + 7 will be moved into address p + 7 and so on until, after moving the byte from address n to address p , X becomes equal to zero and the instruction terminates.

Condition Markers

N	–	*is cleared.*
Z	–	*is set.*
CA	–	*not affected.*
OF	–	*not affected.*

CPBS

0 1	2 … 7	8 … 15
0 0	1 0 0 1 1 1	0 1 0 0 0 0 0 0

Form b (z+x) - b (y+x) Term. if $\not\equiv$

COMPARE BYTE STRINGS: The bytes to be operated on are defined by Y, Z and X as in the instruction MBS. In this case the two strings of bytes are compared with each other. The comparison forms the byte defined by z + x minus the byte defined by y + x. The instruction terminates when X=0 or when two bytes are compared and found to be not equal.

Condition Markers

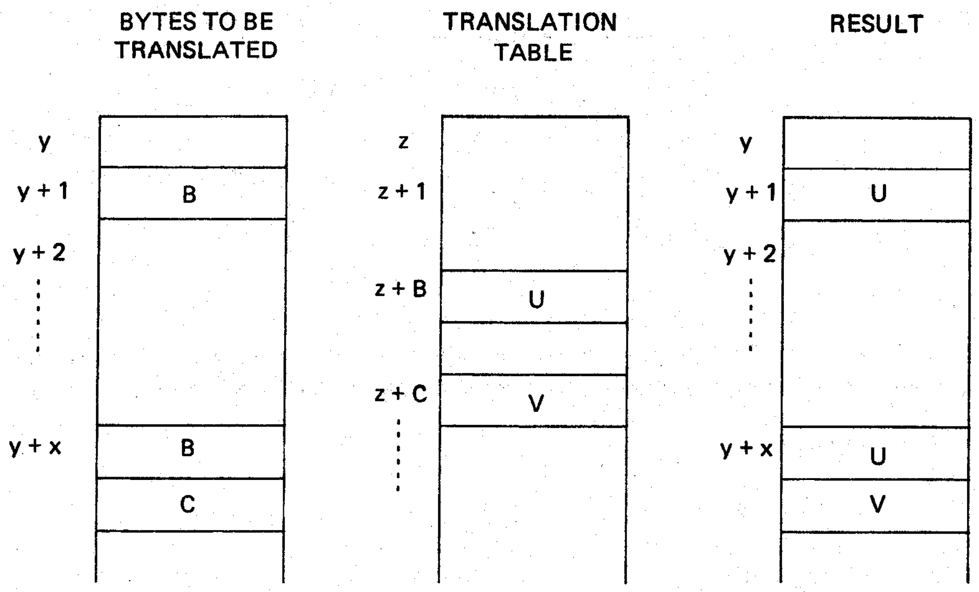
N – *is cleared by the instruction if it runs to completion (i.e. X=0) otherwise it will be set to the sign of the result of the last comparison.*
Z – *is set by the instruction if it runs to completion (i.e. X=0) otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

TRBS

0 1	2 … 7	8 … 15
0 0	1 0 0 1 1 1	1 0 0 0 0 0 0 0

b (y+x) := b (z+b [y+x])

TRANSLATE BYTE STRING: There is a 256 byte translation table beginning at address Z. The string of X bytes, beginning at address Y is examined one byte at a time. Each byte examined is used as an index to select one of the 256 entries in the translation table, so giving the translation for the byte examined. The translation is now written into the place of the byte just examined, so that by the completion of the instruction, every byte in the string has been replaced by its translation.



Condition Markers

N – *is cleared.*
Z – *is set.*
CA – *not affected.*
OF – *not affected.*

SCBS

0 1	2 7	8 15
0 0	1 0 0 1 1 1	1 1 0 0 0 0 0 0

Form b (z+b [y+x])$\wedge$ ls byte of A_L Term. when Result = '0'or when x = '0'

SCAN BYTE STRING: There is a 256 byte table beginning at address Z. The string of X bytes beginning at address Y is examined one byte at a time. Each byte examined is used as an index to select one of the 256 entries in the table, just as for TRBS. The logical AND of the byte from the table and the least significant byte of the accumulator is formed. If the result of this operation is non-zero, the instruction terminates at once, otherwise the instruction continues until the whole string of X bytes has been examined.

This instruction is usually employed to scan a string of characters looking for characters from a specified subset – e.g. any digit. The AND function with the accumulator enables 8 essentially independent sets to be specified using different bit positions within a single 256 byte table. The accumulator usually contains only a single bit.

Condition Markers

N	–	*is cleared.*
Z	–	*is set by the instruction if it runs to completion, otherwise it is cleared.*
CA	–	*not affected.*
OF	–	*not affected.*

MHS

0 1	2 7	8 15
0 0	1 1 0 1 1 1	0 0 0 0 0 0 0 0

h (y+2x) := h (z+2x)

MOVE HALFWORD STRING: The displacement field of this instruction must be zero.

The operation is as for the instruction MBS except that halfwords are moved.

The X register specifies the number of halfwords to be moved and will be scaled to produce halfword addresses.

Condition Markers

N	–	*is cleared.*
Z	–	*is set.*
CA	–	*not affected.*
OF	–	*not affected.*

9.8 MISCELLANEOUS INSTRUCTIONS IN FORMAT L

The following instructions come into this group:–

RK	–	READ KEYS
HRK	–	HALT READ KEYS
SFN	–	SET FULL NUCLEUS
PEC	–	PRIORITY ENCODE
SEXT	–	SIGN EXTEND

RK

0	1	2					7	8							15
0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0

READ KEYS: Information set on the data keys of the CMU front panel is loaded into the least significant 16 bits of the accumulator (A_L). The result in A_L will be sign extended into the most significant 16 bits of the accumulator (A_M). If the CMU is in 'AUTO' mode the 32 bit accumulator will be cleared regardless of the setting of the keys.

Condition Markers

N – *is set to the sign of the 32 bit accumulator after the operation.*
Z – *is set if the 32 bit accumulator becomes zero as a result of the operation, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

HRK

0	1	2					7	8							15
0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1

HALT READ KEYS: If the machine is in TEST or NORMAL mode it will halt before obeying this instruction. Subsequent operation of the START/STOP key on the CMU causes the Read Keys instruction to be obeyed.

With the CMU in 'AUTO' mode the effect of this instruction is as Read Keys.

Condition Markers

N – *is set to the sign of the 32 bit accumulator after the operation.*
Z – *is set if the 32 bit accumulator becomes zero as a result of the operation, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

SFN

0	1	2					7	8							15
0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0

SET FULL NUCLEUS: This instruction switches the computer from Basic Test Mode to Full Nucleus Modes under certain circumstances. It is used to facilitate the loading of programs which will eventually run under full nucleus operation. For the instruction to operate as described the central processor must be in basic test mode with the basic test switch on the front panel in the UP position or Key Switch set to Normal. Under all other conditions the SFN instruction has the same effect as Read Keys.

Condition Markers

N – *is set to the sign of the 32 bit accumulator after the operation.*
Z – *is set if the 32 bit accumulator becomes zero as a result of the operation, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

PEC

0	1	2					7	8							15
0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0

PRIORITY ENCODE: The least significant 16 bits of the accumulator (A_L) are scanned from the most significant bit position to the least significant bit position until the first bit reset to 0 is found. The number of the bit thus found is placed in the X register. The bits of A_L are numbered 0 at the most significant end to 15 at the least significant end. If A_L contains all ones then the number 15 is placed in X as it would be if bit 15 only were set to zero.

Condition Markers

Not affected.

SEXT

0	1	2					7	8							15
0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0

SIGN EXTEND: The sign of the 32 bit accumulator (i.e. the most significant bit) is copied throughout the accumulator extension (B).

Condition Markers

N	–	*is set to the sign of the extended accumulator.*
Z	–	*is set if the extended accumulator becomes zero as a result of the operation, otherwise it is cleared.*
CA	–	*not affected.*
OF	–	*not affected.*

10. INSTRUCTIONS AVAILABLE IN FORMAT RR

These instructions do not require an operand from store, all operations taking place between registers. The operations are defined by the instruction and take place between a source register (G2) and a destination register (G1) both specified by the instruction. These registers may be any two of the following:—

O, A, B, X, L, S, Y, Z,

where O is a non-existent 16 bit register containing zeros. If O is used as a destination register the result of the operation is not recorded but the condition markers are affected. The same register may be both source and destination.

If both G1 and G2 are 16 bit registers then 16 bit operations are performed.

If both G1 and G2 are 32 bit registers then 32 bit operations are performed.

If G1 is a 16 bit register and G2 a 32 bit register then 16 bit operations are performed between G1 and the least significant 16 bits of G2.

If G1 is a 32 bit register and G2 is a 16 bit register then G2 is sign extended to 32 bits and 32 bit operations are performed.

Where G1=S these instructions are effectively branches and with the exception of the load instructions are relative to the next instruction in sequence as for normal branches.

Where G2=S the value of S used for the operation will be that pointing to the next instruction in sequence.

The following instructions are available:—

RLD	–	REGISTER LOAD
RAD	–	REGISTER ADD
RSB	–	REGISTER SUBTRACT
RN	–	REGISTER AND
RCP	–	REGISTER COMPARE
RNA	–	REGISTER NEGATE AND ADD
RO	–	REGISTER OR
RX	–	REGISTER EXCLUSIVE OR
RADC	–	REGISTER ADD PLUS CARRY
RSBC	–	REGISTER SUBTRACT MINUS CARRY
RADI	–	REGISTER ADD PLUS 1
RSBI	–	REGISTER SUBTRACT MINUS 1
RI	–	REGISTER INVERT

RLD

0 5	6 9	10 12	13 15
0 0 0 0 0 0	1 0 0 0	G1	G2

REGISTER LOAD: g1 := g2. The source register specified is loaded into the destination register.

Condition Markers

N – *is set to the sign of the destination register after the operation.*

Z – *is set if the destination register becomes zero as a result of the operation, otherwise it is cleared.*

CA – *not affected.*

OF – *not affected.*

RAD

0 5	6 9	10 12	13 15
0 0 0 0 0 0	1 0 0 1	G1	G2

REGISTER ADD: g1 := g1 + g2. The source register specified is added to the destination register and the result placed in the destination register.

Condition Markers

N – *is set to the sign of the result.*
Z – *is set if the destination register becomes zero as a result of the operation, otherwise it is cleared.*
CA – *is set if there is a carry out of the most significant bit of the result.*
OF – *is set if arithmetic overflow occurs.*

RSB

0 5	6 9	10 12	13 15
0 0 0 0 0 0	1 0 1 0	G1	G2

REGISTER SUBTRACT: g1 := g1 - g2. The source register specified is subtracted from the destination register and the result placed in the destination register.

Condition Markers

N – *is set to the sign of the result.*
Z – *is set if the destination register becomes zero as a result of the operation, otherwise it is cleared.*
CA – *is set if there is a borrow out of the most significant bit of the result.*
OF – *is set if arithmetic overflow occurs.*

RN

0 5	6 9	10 12	13 15
0 0 0 0 0 0	1 1 0 0	G1	G2

REGISTER AND: g1 := g1 ∧ g2. The source and destination registers specified perform the logical AND function the result placed in the destination register.

Condition Markers

N – *is set to the sign of the destination register.*
Z – *is set if the destination register becomes zero as a result of the operation, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

RO

0 5	6 9	10 12	13 15
0 0 0 0 0 0	1 1 0 1	G1	G2

REGISTER OR: g1 := g1 V g2. The source and destination registers specified perform the logical OR function the result placed in the destination register.

Condition Markers

N – *is set to the sign of the destination register.*
Z – *is set if the destination register becomes zero as a result of the operation, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

RCP

0 5	6 9	10 12	13 15
0 0 0 0 0 0	1 0 1 1	G1	G2

REGISTER COMPARE: form g1 - g2. The source register specified is compared with the destination register leaving all registers unchanged.

Condition Markers

N – *is set if the destination register is less than the source register.*
Z – *is set if the two registers being compared are equal, otherwise it is cleared.*
CA – *is set if the result of the comparison produces a borrow out of the most significant bit of the function unit, otherwise it is cleared.*
OF – *is set if arithmetic overflow occurs as a result of the comparison.*

RNA

0 5	6 9	10 12	13 15
0 0 0 0 0 0	0 1 0 0	G1	G2

REGISTER NEGATE AND ADD: g1 := g2 - g1. The destination register specified is subtracted from the source register and the result placed in the destination register.

Condition Markers

N – *is set to the sign of the result.*
Z – *is set if the destination register becomes zero as a result of the operation, otherwise it is cleared.*
CA – *is set if there is a borrow out of the most significant bit of the result.*
OF – *is set if arithmetic overflow occurs.*

RX

0 5	6 9	10 12	13 15
0 0 0 0 0 0	1 1 1 0	G1	G2

REGISTER EXCLUSIVE OR: g1:= g1 ≢ g2. The source register specified and the destination register perform the logical 'EXCLUSIVE OR' function the result being placed in the destination register.

Condition Markers

N – is set to the sign of the destination register.
Z – is set if the destination register becomes zero as a result of the operation, otherwise it is cleared.
CA – not affected.
OF – not affected.

RADC

0 5	6 9	10 12	13 15
0 0 0 0 0 0	0 0 0 1	G1	G2

REGISTER ADD PLUS CARRY: g1 := g1 + g2 + CA. The source register specified is added to the destination register together with the contents of the CARRY condition marker.

Condition Markers

N – is set to the sign of the result.
Z – is set if the destination register becomes zero as a result of the operation, otherwise it is cleared.
CA – is set if there is a carry out of the most significant bit of the result.
OF – is set if arithmetic overflow occurs.

RSBC

0 5	6 9	10 12	13 15
0 0 0 0 0 0	0 0 1 0	G1	G2

REGISTER SUBTRACT MINUS CARRY: g1 := g1 - g2 - CA. The source register is subtracted from the destination register. The contents of the CARRY condition marker is also subtracted from the destination register.

Condition Markers

N – is set to the sign of the result.
Z – is set if the destination register becomes zero as a result of the operation, otherwise it is cleared.
CA – is set if there is a borrow out of the most significant bit of the result.
OF – is set if arithmetic overflow occurs.

RADI

0 5	6 9	10 12	13 15
0 0 0 0 0 0	0 1 0 1	G1	G2

REGISTER ADD PLUS ONE: g1 := g1 + g2 + 1. The source specified is added to the destination register. The integer 1 is also added to the destination register.

Condition Markers

N – *is set to the sign of the result.*
Z – *is set if the destination register becomes zero as a result of the operation, otherwise it is cleared.*
CA – *is set if there is a carry out of the most significant bit of the result.*
OF – *is set if arithmetic overflow occurs.*

RSBI

0 5	6 9	10 12	13 15
0 0 0 0 0 0	0 1 1 0	G1	G2

REGISTER SUBTRACT MINUS ONE: g1 := g1 - g2 - 1. The source register specified is subtracted from the destination register. The integer 1 is also subtracted from the destination register.

Condition Markers

N – *is set to the sign of the result.*
Z – *is set if the destination register becomes zero as a result of the operation, otherwise it is cleared.*
CA – *is set if there is a borrow out of the most significant bit of the result.*
OF – *is set if arithmetic overflow occurs.*

RI

0 5	6 9	10 12	13 15
0 0 0 0 0 0	0 1 1 1	G1	G2

REGISTER INVERT: g1 := $\overline{g2}$: The contents of the source register specified is logically inverted and loaded into the destination register. An alternative equivalent definition is that the integer - 1 is placed in the detination register and the content of the source register is subtracted therefrom.

Condition Markers

N – *is set to the sign of the result.*
Z – *is set if the destination register becomes zero as a result of the operation, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

11. FLOATING POINT OPERATION

The following sections describe the Floating Point features of the computer. Floating Point operations may only be performed if the FM conditions marker is set to a one, when a number of Format A instructions are reinterpreted as Floating Point operations, and a number of additional Format L instructions become available.

11.1 FLOATING POINT NUMBER REPRESENTATION

Two Floating Point number formats are available, differing in the degree of precision they provide. Short precision floating point operands provide 24 bits of mantissa, whilst Long precision operands provide 56 bits of mantissa. In both cases the number is completed by a single bit giving the sign of the mantissa, and a 7 bit exponent.

Short operands therefore occupy 1 (sign)	+ 7 (exponent)	+ 24 (mantissa)	= 32 bits
and Long operands occupy 1	+ 7	+ 56	= 64 bits

In what follows, we define:

M = the arithmetic value of the mantissa of the number, treated as an unsigned fraction.

E = the value of the exponent of the number.

The value of a floating point number is then given by:—

if S = 0, value is $M*16\uparrow E$
if S = 1, value is $-M*16\uparrow E$

Note that a hexadecimal radix is used; increasing E by 1 is equivalent to multiplying M by 16 (i.e. left shifting M four places).

11.2 FLOATING POINT STORE FORMAT

Short floating point numbers are held as fullwords in store, and Long floating point numbers as doublewords in the formats shown:—

Short:—

<table>
<tr><td>0</td><td>1 7</td><td>8 31</td></tr>
<tr><td>S</td><td>EXP</td><td>MANTISSA</td></tr>
</table>

Long:—

<table>
<tr><td>0</td><td>1 7</td><td>8 31</td></tr>
<tr><td>S</td><td>EXP</td><td>MANTISSA (MS 24 Bits)</td></tr>
<tr><td colspan="3">MANTISSA (LS 32 Bits)</td></tr>
</table>

Where S is the sign bit, which determines the sign of the Mantissa as shown above, EXP defines the exponent, whose value is given by E = EXP–64, and MANTISSA is the mantissa, occupying 6 hex digits (24 bits) in short precision or 14 hex digits (56 bits) in long precision formats.

11.3 FLOATING POINT REGISTER FORMAT

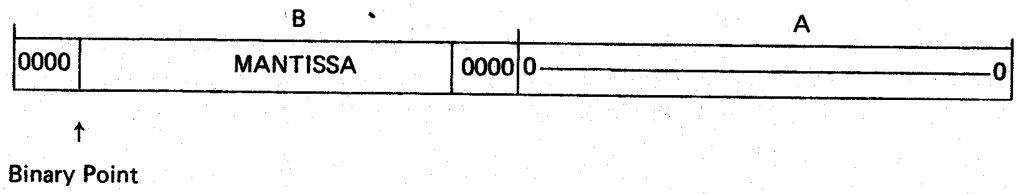
When a floating point operation is performed which leaves a result in registers, the number is held as follows:—

The Sign is held in a hardware staticisor (SAC) which is not directly accessible to the programmer.

The Exponent is held in the 7 bit exponent (E) register.

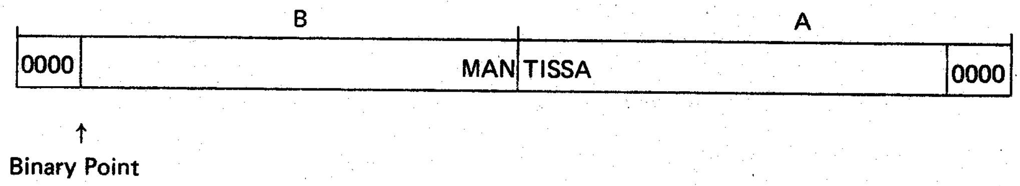
The Mantissa is held in the combined B and A register, the most significant part in the B and the least significant part(if any) in A.

Short mantissae are held as shown:–



The Mantissa is held in B, shifted 4 places. The most significant (MS) and least significant (LS) 4 bits of B are called guard digits, and after any operation will be set to zero. The A register is cleared.

Long mantissae are held as shown:–



The mantissa is held in BA, right shifted 4 places. The MS 4 bits of B and the LS 4 bits of A are the guard digits, and after any operation will be set to zero.

In both cases, the binary point is located immediately before the first hex digit of the Mantissa.

Unpacking of numbers from store to register format and loading and packing of numbers from register to store format on storing are performed automatically.

This structure is referred to as the Floating Accumulator.

11.4 NORMAL REPRESENTATION

A floating point number is said to be normalised if the MS hex digit of the Mantissa is non zero. Thus for a normalised number, M is in the range.

$$1 > M \geqslant 2^{-4}$$

Floating point arithmetic instructions only have defined effects if both operands of the instruction are normalised. Similarly, all arithmetic operations produce results which are normalised.

11.5 FLOATING POINT ZERO

Evidently, any operand with a Mantissa of zero has value zero. The normal representation of zero is an operand with sign bit S and EXP also equal to zero. Thus floating point zero is identical to fixed point zero.

11.6 MIXED PRECISION OPERATION

Short and long floating point operations can be mixed indiscriminately. Thus short floating point operations can be performed with long floating point operands in the accumulator and long operations with short operands in the accumulator. In the first case, the long accumulator operand will be truncated to short precision (by zeroing the least significant 8 hex digits of the mantissa) before the operation, whilst in the second case the short accumulator operand is extended to long precision by appending least significant zero hex digits, before the operation is performed.

11.7 OVERFLOW AND UNDERFLOW

If the result of a floating point operation has an exponent E greater than 63, it cannot be expressed in the normal representation and floating point overflow is said to have occurred. Under these circumstances the OF conditions bit is set, but the result of the operation is not defined.

If the result of a floating point operation has an exponent E less than –64, or if the mantissa of a result is zero, underflow is said to have occurred. This is not an error: the result is replaced by floating point zero in standard form.

12. THE FLOATING POINT INSTRUCTION SET

The following instructions change their definition when the FM condition marker is set:–

FM = 0	FM = 1		
LD	FLD	–	FLOATING LOAD
AD	FAD	–	FLOATING ADD
SB	FSB	–	FLOATING SUBTRACT
CP	FCP	–	FLOATING COMPARE
N	FLT	–	FLOAT
M	FM	–	FLOATING MULTIPLY
D	FD	–	FLOATING DIVIDE
ST	FST	–	FLOATING STORE
LDW	ELD	–	EXTENDED FLOATING LOAD
ADW	EAD	–	EXTENDED FLOATING ADD
SBW	ESB	–	EXTENDED FLOATING SUBTRACT
CPW	ECP	–	EXTENDED FLOATING COMPARE
NW	FIX	–	FIX
MW	EM	–	EXTENDED FLOATING MULTIPLY
DW	ED	–	EXTENDED FLOATING DIVIDE
STW	EST	–	EXTENDED FLOATING STORE

12.1 NORMAL LENGTH OPERATIONS

These operations all use short Floating Point operands.

FLD

0 5	6 15	
1 0 0 0 0 0	ADDR	(FM = 1 only)

FLOATING LOAD: fa := fq. The short floating point operand from store is loaded into the floating accumulator.

No normalisation checks are performed during this instruction; thus it is possible to load an unnormalised number into the registers. If the mantissa of such a number is zero, the Z condition bit is set even if S and EXP are not zero.

Condition Markers

N – *is set to the sign of the floating point number loaded.*
Z – *is set if the floating point number loaded is zero, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

FAD

0 5	6 15	
1 0 0 0 0 1	ADDR	(FM = 1 only)

FLOATING ADD: fa := fa + fq. The short floating point operand from store is added to the short floating point number held in the Floating Accumulator and the short result appears in the Floating Accumulator. Details of rounding etc. are given in Appendix 12A.

Condition Markers

N	–	*set if the result is negative.*
Z	–	*set if the result is zero, or if exponent underflow occurs.*
CA	–	*not affected.*
OF	–	*set to 1 if exponent overflow (EXP > 127) occurs.*

FSB

0 … 5	6 … 15	
1 0 0 0 1 0	ADDR	(FM = 1 only)

FLOATING SUBTRACT: fa := fa – fq. The short floating point operand from store is subtracted from the short floating point number held in the Floating Accumulator and the short result appears in the Floating Accumulator. Details of rounding etc. are given in Appendix 12A.

Condition Markers

N	–	*set if the result is negative.*
Z	–	*set if the result is zero, or if exponent underflow occurs.*
CA	–	*not affected.*
OF	–	*set to 1 if exponent overflow (EXP > 127) occurs.*

FCP

0 … 5	6 … 15	
1 0 0 0 1 1	ADDR	(FM = 1 only)

FLOATING COMPARE: form fa – fq. The short floating point number held in the Floating Accumulator is compared with the short floating point operand from store. The condition markers indicate the result.

Condition Markers

N	–	*is set if the floating point number held in the Floating Accumulator is less than the floating point operand from store, otherwise it is cleared.*
Z	–	*is set if the floating point number held in the Floating Accumulator is equal to the floating point operand from store, otherwise it is cleared.*
CA	–	*not affected.*
OF	–	*not affected.*

FM

0 … 5	6 … 15	
1 0 0 1 0 1	ADDR	(FM = 1 only)

FLOATING MULTIPLY: ea := fa * fq. The short floating point number held in the registers (the multiplicand) is multiplied by the short floating point operand from store,(the multiplier). The 48 bit product is extended to 56 bits with least significant zeros to produce a long floating point result which is normalised and placed in the Floating Accumulator. Details of the operation are given in Appendix 12A.

Condition Markers

N	–	*is set to the sign of the floating point result.*
Z	–	*is set if the floating point result is zero or if underflow occurs as above. If neither of these conditions apply Z is cleared.*
CA	–	*not affected*
OF	–	*is set if exponent overflow occurs in which case the result, N and Z are undefined.*

FD

0 5	6 15
1 0 0 1 1 0	ADDR

(FM = 1 only)

FLOATING DIVIDE: fa := fa ÷ fq. The short floating point number held in the Floating Accumulator (the dividend) is divided by the short floating point operand from store (divisor). The short result appears in the Floating Accumulator. Details of the operation are given in Appendix 12A.

If an attempt is made to divide with a zero or unnormalised divisor the operation is abandoned. The OF condition marker is set and the result is undefined. The least significant 32 bits of the extended accumulator (A_M and A_L) will not be cleared in this case.

Condition Markers

N – *is set to the sign of the floating point result.*
Z – *is set if the mantissa of the floating point result is zero or if undeflow occurs as above. If neither of these conditions apply Z is cleared.*
CA – *not affected.*
OF – *is set if exponent overflow occurs in which case the result, N and Z are undefined.*

FST

0 5	6 15
1 0 0 1 1 1	ADDR

(FM = 1 only)

FLOATING STORE: fq := fa. The short floating point operand in the Floating Accumulator registers are stored in the format shown in 8.2.

Condition Markers

N – *is set to the sign of the stored floating point number.*
Z – *is set if the floating point number is zero, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

12.2 EXTENDED OPERATIONS

These operations all use long floating point operands.

ELD

0 5	6 15
1 0 1 0 0 0	ADDR

(FM = 1 only)

EXTENDED LOAD: ea := eq. The long floating point operand from store is loaded into the Floating Accumulator. Normalisation checks are NOT carried out during this instruction. It is thus possible to load unnormalised operands into the registers. If the mantissa of such a number is zero, the Z condition bit is set even if S and EXP are non zero.

Condition Markers

N – *is set to the sign of the extended point number loaded.*
Z – *is set if the floating point number loaded is zero, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

EAD

0 5	6 15
1 0 1 0 0 1	ADDR

(FM = 1 only)

EXTENDED ADD: ea := ea + eq. The long floating point operand from store is added to the long floating point number held in the Floating Accumulator and the result appears in the Floating Accumulator.

Condition Markers

N – *set if the result is negative.*
Z – *set if the result is zero, or if exponent underflow occurs.*
CA – *not affected.*
OF – *set to 1 if exponent overflow (EXP > 127) occurs.*

ESB

0 5	6 15
1 0 1 0 1 0	ADDR

(FM = 1 only)

EXTENDED SUBTRACT: ea := ea - eq. The long floating point operand from store is subtracted from the long floating point number held in the Floating Accumulator and the result appears in the Floating Accumulator.

Condition Markers

N – *set if the result is negative.*
Z – *set if the result is zero, or if exponent underflow occurs.*
CA – *not affected.*
OF – *set to 1 if exponent overflow (EXP > 127) occurs.*

ECP

0 5	6 15
1 0 1 0 1 1	ADDR

(FM = 1 only)

EXTENDED COMPARE: form ea - eq. The long floating point number held in the Floating Accumulator compared with the long floating point operand from store. The condition markers indicate the result.

Condition Markers

N – *is set if the floating point number held in the Floating Accumulator is less than the floating point operand from store, otherwise it is cleared.*
Z – *is set if the floating point number held in the Floating Accumulator is equal to the floating point operand from store, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

EM

0 5	6 15
1 0 1 1 0 1	ADDR

(FM = 1 only)

EXTENDED MULTIPLY: ea := ea ∗ eq. The long floating point number held in the Floating Accumulator (the multiplicand) is multiplied by long floating point operand from store (the multiplier). The long floating point result is placed in the Floating Accumulator.

Condition Markers

N – *is set to the sign of the floating point result.*
Z – *is set if the floating point result is zero or if underflow occurs as above. If neither of these conditions apply Z is cleared.*
CA – *not affected.*
OF – *is set if exponent overflow occurs in which case the result, N and Z are undefined.*

ED

0 5	6 15
1 0 1 1 1 0	ADDR

(FM = 1 only)

EXTEND DIVIDE: ea := ea ÷ eq. The long floating point number in the Floating Accumulator (the dividend) is divided by the long floating point operand from store (the divisor). The long result is placed in the Floating Accumulator.

Condition Markers

N – *is set to the sign of the floating point result.*
Z – *is set if the mantissa of the floating point result is zero or if underflow occurs as above. If neither of these conditions apply Z is cleared.*
CA – *not affected*
OF – *is set if exponent overflow occurs in which case the result, N and Z are undefined.*

EST

0 5	6 15
1 0 1 1 1 1	ADDR

(FM = 1 only)

EXTENDED STORE: eq := ea. The extended floating point operand in the Floating Accumulator is stored in the format shown in 8.2.

Conditions Markers

N – *is set to the sign of the stored floating point number.*
Z – *is set if the floating point number is zero, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

12.3 FLOATING POINT CONVERSION INSTRUCTIONS

FLT

0 5	6 15
1 0 0 1 0 0	ADDR

(FM = 1 only)

FLOAT AND LOAD: A 32 bit integer from store is converted into the equivalent long floating point number and loaded into the Floating Accumulator. The operation is described in detail in Appendix 12A.

Condition Markers

N – *is set to the sign of the floating point number.*
Z – *is set if the floating point number generated is zero, otherwise it is cleared.*
CA – *not affected.*
OF – *not affected.*

FIX

0 5	6 15
1 0 1 1 0 0	ADDR

(FM = 1 only)

FIX AND STORE: The long floating point number in the Floating Accumulator converted into an equivalent integer and stored. A fractional floating point remainder is left in the Floating Accumulator.

If the original Floating point number is in the range $-1 < \text{Number} < +1$, the result is zero, and the Floating Accumulator is unchanged. If the number is outside the range $-2^{31} \leqslant \text{Number} \leqslant 2^{31} - 1$, overflow will occur. Details of the operation are given in Appendix 12A.

Condition Markers

N – *is set to the sign of the integer that is stored.*
Z – *is set if the integer formed is zero, otherwise it is cleared.*
CA – *not affected.*
OF – *is set if significant bits are lost when truncating the integer formed to 32 bits, i.e. if the floating point number to be fixed is unable to be represented as a 32 bit integer and fractional floating point remainder.*

12.4 FLOATING POINT INSTRUCTIONS IN FORMAT L

There are 3 such instructions:–

SFM – SET FLOATING MODE
SIM – SET INTEGER MODE
FNEG – NEGATE FLOATING POINT NUMBER

As described in section 5 the CPU has an integer mode of operation and a floating point mode of operation. The instructions SFM and SIM provide the facility for switching between these two modes of operation.

If the instruction FNEG is used with the central processor in integer mode the effect is undefined.

SFM

0 1	2 7	8 15
0 0	0 1 0 1 1 1	0 0 0 0 0 0 0 1

SET FLOATING MODE: FM := 1. When executed this instruction causes the floating marker flag (FM) to be set. This flag is part of the register that holds the condition markers and is treated in the same way as N, Z, OF and CA.

If this instruction is executed with the FM flag already set then it has no effect.

Condition Markers

Not affected.

SIM

0 1	2 7	8 15
0 0	0 1 0 1 1 1	0 0 0 0 0 0 0 0

SET INTEGER MODE: FM = 0. When executed this instruction causes the floating marker flag (FM) to be cleared.

If the instruction is executed with FM already cleared then it has no effect.

Condition Markers

Not affected.

FNEG

0 1	2 7	8 15	
0 0	0 1 0 1 1 1	0 0 0 0 0 0 1 0	(FM = 1 only)

FLOATING NEGATE: ea := -ea. When executed this instruction changes the sign of the floating point number held in the Floating Accumulator. If the instruction is executed with FM=0, it will affect the A register in an undefined manner.

If the floating point number held in the Floating Accumulator is not normalised the instruction will cause it to be replaced with floating point zero.

Condition Markers

N	–	*is set to the new sign of the floating point number held in the register.*
Z	–	*is set if the result is zero, otherwise it is cleared.*
CA	–	*not affected.*
OF	–	*not affected.*

APPENDIX 12A: FLOATING POINT ARITHMETIC

1.1 INTRODUCTION

The following sections describe the operations of Floating Point Addition, Subtraction, Multiplication, and Division. They are provided for the benefit of those who require to know details of accuracy, rounding etc. It should be noted that the algorithms presented are idealised, and have the same effect as but are not necessarily the same as the algorithms implemented by the hardware. For instance the hardware does not perform the arbitrarily long shifts implied in some places.

1.2 FLOATING POINT ADDITION AND SUBTRACTION

Floating Point Addition is carried out as described in this section. Floating Point Subtraction is carried out by negating the subtrahend (by logically inverting its sign bit if it is non zero) and then performing an addition.

(a) The exponents of the two operands are compared. If they differ, the operand with the smaller exponent is aligned by shifting its Mantissa right four places at a time, incrementing its exponent by 1 for each four places shifted, until the two exponents are equal. After each shift, the four bits just shifted out of the mantissa are retained in the Guard Digit. If the number of places of shift required is greater than 28 (short precision) or 60 (long precision), then the operand will be shifted completely out of the accumulator. In this case the result is equal to the larger of the two operands.

(b) An intermediate result is now formed whose exponent is equal to the larger of the two operand exponents, and whose sign and (positive) mantissa are found by adding or subtracting the two mantissae having regard for their signs and magnitudes.

If the signs of the operands are the same, the mantissae are added, and the sign of the result is the same as the sign of the operands.

If the signs differ, the smaller mantissa is subtracted from the larger mantissa, and the sign of the result is the same as the sign of the larger operand.

In all cases the mantissa of the intermediate result consists of 32 bits (short precision) or 64 bits (long precision) and is in the range $0 \leqslant$ mantissa < 2.

(c) The final result is now formed by normalising the intermediate result if necessary as follows:—

(i) If the mantissa is already normalised, no further operations are necessary, and the final result equals the intermediate result with its guard digit reset to zero.

(ii) If the intermediate result mantissa is zero, then a clean zero result is generated by setting the sign and exponent to zero.

(iii) If the intermediate result mantissa is greater than or equal to 1, it is normalised by right shifting four places and adding one to the exponent. The guard digit is then reset to zero. If after the shift the exponent is greater than 63, overflow has occurred. OF is set and the result is undefined.

(iv) If the mantissa is less than $1/16$, it is left shifted four places at a time until it is normalised, subtracting 1 from the exponent for each four places shifted. The guard digit is then reset to zero. If after the shifts, the exponent is less than -64, underflow has occurred. In this case the result is replaced by clean zero.

1.3 FLOATING POINT MULTIPLICATION

Floating Point Multiplication is carried out as follows:

(a) An intermediate result is formed by adding the exponents of the operands and multiplying their mantissae. The sign of the result is positive if the signs of the operands are the same, and is negative otherwise. The mantissa is in the range

$$\frac{1}{256} \leqslant \text{mantissa} < 1$$

and for long operation is the most significant 60 bits of the 112 bit product, whilst for short operations the 48 bit product is extended to 60 bits by appending low order zeroes. The exponent of the intermediate result is held to 8 bits, and no overflow can occur at this stage.

(b) The final result is now formed by normalising the intermediate result if necessary as follows:—

(i) If the mantissa is already normalised, no further operations are necessary. The result is truncated to 56 bits by zeroing the guard digit.

(ii) If the intermediate result mantissa is zero, then a clean zero result is generated.

(iii) If the mantissa is less than $1/16$, it is normalised by left shifting four places and subtracting 1 from the exponent. Note that only one normalising shift is necessary. The result is then truncated to 56 bits as described above.

(c) If after normalisation, the exponent is greater than 63, overflow has occurred. OF is set and the result is undefined. If after normalisation the exponent is less than -64, underflow has occurred. The result is replaced by clean zero.

(d) Note that the result of a short Floating Point multiply will be a long Floating Point number, and will not be truncated to 24 bits.

1.4 FLOATING POINT DIVISION

Floating Point Division is carried out as follows:

(a) If the divisor (store operand) is zero, the operation cannot be performed. OF is set to record overflow, and the result is undefined.

(b) Otherwise, an intermediate result is formed by subtracting the exponent of the divisor from the exponent of the dividend (accumulator operand) and dividing the mantissa of the dividend by the mantissa of the divisor. The sign of the result is positive if the sign of the operands is the same, and is negative otherwise. The mantissa is in the range

$$\frac{1}{16} < \text{mantissa} < 16$$

and comprises the ms 28 bits of the quotient (short precision) or 60 bits (long precision). The exponent is held to 8 bits and no overflow can occur at this stage.

(c) The final result is now formed by normalising the intermediate result if necessary as follows:—

(i) If the mantissa is already normalised, no further operations are necessary. The result is truncated to 24 bits (short operations) or 56 bits (long operations) by zeroing the guard digit.

(ii) If the intermediate result mantissa is zero (this can only occur if the dividend was zero) then a clean zero result is generated.

(iii) If the mantissa is greater than or equal to 1, it is normalised by right shifting four places and adding 1 to the exponent. Note that only one normalising shift is necessary. The result is then truncated to 24 bits (short) or 56 bits (long) as described above.

(d) If, after normalisation, the exponent is greater than 63, overflow has occurred. OF is set and the result is undefined.

If after normalisation the exponent is less than -64, underflow has occurred. The result is replaced by clean zero.

1.5 THE FLT INSTRUCTION

The Float operation is carried out as follows:-

A long floating point number is formed with E = +8, MANTISSA = magnitude of the integer operand, and S = 0 if the integer is positive or zero, and S = 1 if the integer is negative. This long floating point number is now normalised if necessary and placed in the Floating Accumulator. If its mantissa is zero, a clean Zero result is entered.

1.6 THE FIX INSTRUCTION

The Fix operation is carried out as follows:-

First the exponent E of the number in the Floating Accumulator is tested. If E ⩽ 0, the number has no integer part. In this case a zero integer is placed in store and the Floating Point accumulator is unchanged; this completes the operation.

If E ⩾ 1, the number can be expressed as either + (I + F) if it is positive, or - (I + F) if it is negative, where in both cases I is a non zero integer and F is a (possibly zero) fraction. The 2s complement integer result in store is formed by taking I, negating it if it was negative, and truncating to 32 bits. If significant bits are lost in the truncation, Overflow has occurred, and the result placed in store will be the LS 32 bits of the true integer result.

Whether overflow occurs or not, the fractional remainder F is found by left shifting the mantissa of the original number four places at a time, decrementing E by 1 for each place of shift, until E becomes zero. The resulting number is now normalised if necessary and placed in the Floating Accumulator. If its mantissa is zero, a clean zero result is recorded.

Note that the fractional remainder will always be correct, even if the integer result overflows.

13. CONTROL INSTRUCTIONS AVAILABLE IN BASIC MODE

Channel Interrupt

Eight interrupt lines are provided on the Command Interface (see CPU Basic Multiplexer Channel Manual) and any one of eight Input Output Processors (including the Basic Multiplexer Channel) may interrupt over this interface.

When in basic test mode the central processor takes the following action if interrupt by any input output processor:–

(i) The number of the highest priority channel interrupting the processor is written into the halfword location at address zero.

(ii) The contents of the program accessible registers S, L, B, A, X, Y, Z, E/C are written into locations 2 onwards of store.

STORE LAYOUT

0	=	INT CODE (ERROR CODE)
2	=	S Register
4	=	L Register
6	=	BM Register
8	=	BL Register
10	=	AM Register
12	=	AL Register
14	=	X Register
16	=	Y Register
18	=	Z Register
20	=	E/C Register
22	=	INTERRUPT VALUE OF S
24	=	INTERRUPT VALUE OF L

(iii) The S register is loaded from location 22 of store and the L register from location 24.

(iv) Interrupts are inhibited.

Error Interrupts

If an error condition arises whilst a program is running the following sequence of events is performed:–

(i) An integer value is placed in location 0 of store to identify the cause of the interrupt.

(ii) The interrupt sequence in the previous paragraph is performed.

ERROR CODES

INT CODE ERROR	CODE
Store Parity	8
Power Fail	9
Store Time Out	12
Start Up	13
Illegal Instruction	14
Command Interface Time Out	16

NOTE: When starting from the RESET condition the S and L registers are loaded from locations 22 and 24 respectively and code 13 is placed in location 0.

The following instructions are provided in Format L for the control of input/output and interrupt handling in Basic Test Mode:–

INT	–	INTERRUPT CONTROL
I/O	–	INPUT/OUTPUT CONTROL

13.1 INT – INTERRUPT CONTROL

Five variants of this instruction are provided and these are further defined by the displacement field of the instruction as follows:–

8 12	13 15
0 0 0 0 0	M

M	INSTRUCTION		
000	AINT	–	ACKNOWLEDGE INTERRUPT
001	PERM	–	PERMIT INTERRUPT
010	INH	–	INHIBIT INTERRUPT
011	TERM	–	TERMINATE INTERRUPT PROGRAM
100	SINT	–	SOFTWARE INTERRUPT

Interrupt Instructions

AINT

0 1	2 7	8 15
0 0	0 1 1 1 0 0	0 0 0 0 0 0 0 0

ACKNOWLEDGE INTERRUPT: This instruction allows an acknowledge interrupt cycle to take place over the command interface causing the selected IOP to reset its interrupt line.

The number of the IOP involved is held in location zero of store.

Condition Markers

Not affected.

PERM

0	1	2					7	8							15
0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1

PERMIT INTERRUPTS: This instruction allows the central processor to take the action described in Channel Interrupt on receipt of an interrupt.

Condition Markers

Not affected.

INH

0	1	2					7	8							15
0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0

INHIBIT INTERRUPTS: This instruction prevents the central processor from taking the action described in Channel Interrupt until after a PERM or TERM instruction has been executed.

Condition Markers

Not affected.

TERM

0	1	2					7	8							15
0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	1

TERMINATE INTERRUPT PROGRAM: This instruction causes the program accesible registers S, L, B, A, X, Y, Z, E/C to be loaded from location 2 onwards of store. Having loaded the registers interrupts are then permitted.

Condition Markers

The Condition Markers are loaded from byte 21 of store.

SINT

0	1	2					7	8							15
0	0	0	1	1	1	0	0	0	0	0	0	0	1	0	0

GENERATE SOFTWARE INTERRUPT: The effect of this instruction is to store register A_L in location zero of store and then force the central processor to take action as if it had received an interrupt (i.e. that described in Channel interrupt). This instruction takes effect whether interrupts are inhibited or not.

Condition Markers

Not affected.

13.1 I/O – INPUT/OUTPUT CONTROL

Two variants of this instruction are provided and these are further defined by the displacement field of the instruction as follows:–

8	9	10 12	13 15
0	M	0 0 0	N

The N field of the instruction defines the channel number over which input/output is to be carried out, i.e. channels 0–7.

The M field defines whether input or output is to take place.

M	ACTION
0 1	IN – INPUT OVER CHANNEL DEFINED BY N OUT – OUTPUT OVER CHANNEL DEFINED BY N

IN

0 1	2 7	8 12	13 15
0 0	0 1 1 1 0 1	0 0 0 0 0	N

INPUT FROM CHANNEL N: The way number of the selected device is defined by the contents of the Z register.

Data from the selected device is placed in A_L with A_M being cleared. The condition markers are set to indicate the result of the operation as follows:–

N	Z	OF	CA	MEANING
0	0	0	0	Operation Successfull
X	X	1	X	Operation Failed

When an error is indicated by OF being set to 1, N,Z and CA may be used to indicate the cause of failure by the IOP.

OUT

0 1	2 7	8 12	13 15
0 0	0 1 1 1 0 1	0 1 0 0 0	N

OUTPUT TO CHANNEL N: The way number of the selected device is defined by the contents of register Z as it is for input.

Data from A_L is output to the selected device at the same time A_M is cleared. Data from the IOP may be input to A_L at this time, however, the BMC does not make use of this feature and thus A_L is left with the original data that has been output to the IOP.

The condition markers are set to indicate the result of the operation as for input.

Supplement 1: SUMMARY OF INSTRUCTIONS

SUMMARY OF INSTRUCTION FORMATS

This Supplement provides a quick tabular reference to the instructions available in the computer instruction set. A full description of each instruction, and the formats in which they are available, is given earlier in this manual.

FORMAT B

0 5	6 15
F	D

Q = S ± 2D (D treated as a Signed Integer)

FORMAT A1

0 5	6 7	8 15
F	0 1	D

$Q = D^*$

FORMAT A2

0 5	6	7 8	9 15
F	1	M	D

M = 0	$Q = I + D^*$
= 1	$= s + D^*$
= 2	$= y + D^*$
= 3	$= z + D^*$

FORMAT A3

0 5	6 9	10 15
F	0 0 0 1	D

$Q = (2D) + x^*$

FORMAT A4

0 5	6 8	9 10	11 15
F	0 0 1	M	D

M = 0	$Q = (I + 2D) + x^*$
= 1	$= (s + 2D) + x^*$
= 2	$= (y + 2D) + x^*$
= 3	$= (z + 2D) + x^*$

FORMAT A5

0 5	6 9	10 11	12 15
F	0 0 0 0	M	D

M = 0	Q = I + D* + x*
= 1	= D* + x*
= 2	= y + D* + x*
= 3	= z + D* + x*

FORMAT L

0 1	2 7	8 15
0 0	F	D

Arithmetic Operations – D is unsigned literal operand
Conditional Branches – D is signed Integer Displacement
Control Operations – D defines operation
Shifts, Bit Operations – See below.

FORMAT RR

0 5	6 9	10 12	13 15
0 0 0 0 0 0	F	G1	G2

G1, G1 = 0	Register = 0
= 1	= A
= 2	= B
= 3	= X
= 4	= L
= 5	= S
= 6	= Y
= 7	= Z

FORMAT L – SHIFTS

0 7	8 10	11 15
SHL/SHX	M	N

M defines shift type
N defines places of shift = N (left shifts)
= 32–N (right shifts)

FORMAT L – BIT OPS

0 7	8 11	12 15
BITL/BITX	M	N

M defines operation
N defines bit number of l.s. 16 bits of A involved

SUMMARY OF INSTRUCTIONS

MNEMONIC	EFFECT	CONDITIONS				HEX	FORMAT	CLASS
		N	Z	OF	CA			
LD*	a := hq	a <0	a=0	–	–	8000	A	H
AD*	a := a + hq	r<0	a=0	r	r	8400	A	H
SB*	a := a - hq	r <0	a=0	r	r	8800	A	H
CP*	form a - hq and set N,Z,CA,OF	a<hq	a=hq	r	r	8C00	A	H
N*	a := a ∧ hq	a<0	a=0	–	–	9000	A	H
M*	a := a * hq	r<0	a=0	r	–	9400	A	H
D*	a := a ÷ hq; b := remainder	a<0	a=0	r	–	9800	A	H
ST*	hq := ha	hq<0	hq=0	hq≢a	–	9C00	A	H
LDW*	a := wq	a< 0	a=0	–	–	A000	A	W
ADW*	a := a + wq	r< 0	a=0	r	r	A400	A	W
SBW*	a := a– wq	r< 0	a=0	r	r	A800	A	W
CPW*	form a– wq and set N,Z,CA,OF	a< wq	a = wq	r	r	AC00	A	W
NW*	a := a ∧ wq	a<0	a=0	–	–	B000	A	W
MW*	da := a * wq	r<0	da=0	–	–	B400	A	W
DW*	a := da ÷ wq; b := remainder	r <0	a= 0	–	–	B800	A	W
STW*	wq := a	wq<0	wq=0	–	–	BC00	A	W
LDX	x := hq	x<0	x=0	–	–	C000	A	H
ADX	x := x + hq	r<0	x=0	r	r	C400	A	H
SBX	x := x - hq	r<0	x=0	r	r	C800	A	H
CPX	form x - hq and set N,Z,CA,OF	x<hq	x=hq	r	r	CC00	A	H
NX	x := x ∧ hq	x<0	x=0	–	–	D000	A	H
MX	x := x* hq	r<0	x=0	r	–	D400	A	H
DX	x := x ÷ hq	x<0	x=0	r	–	D800	A	H
STX	hq := x	hq<0	hq=0	–	–	DC00	A	H
LDY	y := hq	–	–	–	–	E000	A	H
ADY	y := y + hq	–	–	–	–	E400	A	H
SBY	y := y - hq	–	–	–	–	E800	A	H
STY	hq := y	–	–	–	–	EC00	A	H
LDZ	z := hq	–	–	–	–	F000	A	H
ADZ	z := z + hq	–	–	–	–	F400	A	H
SBZ	z := z - hq	–	–	–	–	F800	A	H
STZ	hq := z	–	–	–	–	FC00	A	H
LDB	a := bq	0	a=0	–	–	4000	A	B
LBX	x := bq	0	x=0	–	–	4400	A	B
CPB	form a – bq and set N,Z,CA,OF	a<bq	a=bq	–	–	4C00	A	B
NBS	bq := bq ∧ ba	0	bq=0	–	–	5000	A	B
OBS	bq := bq ∨ ba	0	bq=0	–	–	5400	A	B
XBS	bq := bq ≢ ba	0	bq=0	–	–	5800	A	B
STB	bq := ba	0	bq=0	–	–	5C00	A	B

* Instructions available if FM = 0 only.

MNEMONIC	EFFECT	CONDITIONS				HEX	FORMAT	CLASS
		N	Z	OF	CA			
DECS	hq := hq - 1	r<0	hq=0	r	r	4800	A	H
INCS	hq := hq + 1	r<0	hq=0	r	r	7C00	A	H
HAY	y := hQ	–	–	–	–	6000	A	H
HAZ	z := hQ	–	–	–	–	7000	A	H
LDM	load B,A,X,Y,Z,E,C from WQ:WQ+14	C_4*	C_5*	C_6*	C_7*	6400	A	W
STM	store B,A,X,Y,Z,E,C in WQ: WQ+14	–	–	–	–	6800	A	W
LDL	a := D	0	a=0	–	–	2000	L	LB
ADL	a := a + D	r<0	a=0	r	r	2100	L	LB
SBL	a := a - D	r<0	a=0	r	r	2200	L	LB
CPL	form a - D and set N,Z,CA,OF	a<D	a=D	r	r	2300	L	LB
NL	a := a ∧ D	0	a=0	–	–	2400	L	LB
ML	a := a * D	r<0	a=0	r	–	2500	L	LB
DL	a := a ÷ D; b := remainder	a<0	a=0	r	–	2600	L	LB
LDXL	x := D	0	x=0	–	–	3000	L	LB
ADXL	x := x + D	r<0	x=0	r	r	3100	L	LB
SBXL	x := x - D	r<0	x=0	r	r	3200	L	LB
CPXL	form x - D and set N,Z,CA,OF	x<D	x=D	r	r	3300	L	LB
NXL	x := x ∧ D	0	x=0	–	–	3400	L	LB
MXL	x := x *D	r<0	x=0	r	–	3500	L	LB
DXL	x := x ÷ D	x<0	x=0	r	–	3600	L	LB
LDYL	y := D	–	–	–	–	3800	L	LB
ADYL	y := y + D	–	–	–	–	3900	L	LB
SBYL	y := y - D	–	–	–	–	3A00	L	LB
CPYL	form y - D and set N,Z,CA,OF	y<D	y=D	r	r	1B00	L	LB
LDZL	z := D	–	–	–	–	3F00	L	LB
ADZL	z := z + D	–	–	–	–	3D00	L	LB
SBZL	z := z - D	–	–	–	–	3E00	L	LB
CPZL	form z - D and set N, Z,CA,OF	y<D	y=D	r	r	1F00	L	LB
B	s := Q	–	–	–	–	0400	B	
BL	z := s ; s := Q	–	–	–	–	0800	B	
BI	s :=hq	–	–	–	–	7400	A	H
BLI	z := s ; s := hq	–	–	–	–	7800	A	H
BN	s := s ± D if N = 1	–	–	–	–	2800	L	BR
BNN	s := s ± D if N = 0	–	–	–	–	2900	L	BR
BZ	s := s ± D if Z = 1	–	–	–	–	2A00	L	BR
BNZ	s := s ± D if Z = 0	–	–	–	–	2B00	L	BR
BP	s := s ± D if N = 0 & Z = 0	–	–	–	–	2C00	L	BR
BNP	s := s ± D if N = 1 or Z = 1	–	–	–	–	2D00	L	BR
BOF	s := s ± D if OF = 1 ; OF = 0	–	–	0	–	2E00	L	BR
BNCA	s := s ± D if CA = 0	–	–	–	–	2F00	L	BR
BPAR	s := s ± D if ba has odd parity	–	–	–	–	1800	L	BR

* $C_{4,5,6,7}$ are the LS 4 bits of byte WQ +14.

MNEMONIC	EFFECT	CONDITIONS				HEX	FORMAT	CLASS
		N	Z	OF	CA			
SHL	Shift Literal: D defines type and places					19XX	L	SH
SHX	Shift Indexed: D + x defines type and places.					3BXX	L	SH
	(N is l.s. 5 bits of D + x)							
SBAR,SBRX	Shift da right arithmetic 32–N places	da<0	da=0	–	–	XX00	L	SH
SBAL,SBLX	Shift da left arithmetic N places	da<0	da=0	r	–	XX20	L	SH
SR, SRX	Shift a right arithmetic 32–N places	a<0	a=0	–	–	XX40	L	SH
SL, SLX	Shift a left arithmetic N places	a<0	a=0	r	–	XX60	L	SH
SRL, SRLX	Shift a right logical 32–N places	a<0	a=0	–	–	XX80	L	SH
SLC, SLCX	Shift a left circular N places	a<0	a=0	–	–	XXA0	L	SH
SXR, SXRX	Shift x right arithmetic 32–N places	x<0	x=0	–	–	XXD0	L	SH
SXL, SXLX	Shift x left arithmetic N places	x<0	x= 0	–	–	XXE0	L	SH
BITL	Bit operation Literal: D defines operation					16XX	L	BIT
BITX	Bit operation Indexed: D + x defines operation					15XX	L	BIT
	(N is l.s. 4 bits of D + x)						L	BIT
TSTB,TSTX	Set Z = 1 if bit N of ha is zero		bit=0	–	–	XX00	L	BIT
TGLB,TGLX	Change the state of bit N of ha		ha=0	–	–	XX90	L	BIT
PLCB,PLCX	Set bit N of ha to 1, reset all other bits		0	–	–	XXA0	L	BIT
SETB,SETX	Set bit N of ha to 1		ha=0	–	–	XXB0	L	BIT
CLRB,CLRX	Reset bit N of ha to 0		ha=0	–	–	XXE0	L	BIT
MBS	Move byte strings	0	1	–	–	2700	L	C
CPBS	Compare byte strings	r<0	end	–	–	2740	L	C
TRBS	Translate byte strings	0	1	–	–	2780	L	C
SCBS	Scan byte strings	0	end	–	–	27C0	L	C
MHS	Move halfword strings	0	1	–	–	3700	L	C
RK	a := keys	a<0	a=0	–	–	1200	L	C
HRK	Halt processor; then a := keys	a<0	a=0	–	–	1201	L	C
PEC	x := priority encode of bits of ha	–	–	–	–	1E00	L	C
SEXT	da := a	da<0	da=0	–	–	3C00	L	C
RADC	g1 := g1 + g2 + CA	r<0	g1=0	r	r	0040	RR	
RSBC	g1 := g1 - g2 - CA	r<0	g1=0	r	r	0080	RR	
RNA	g1 := g2 - g1	r<0	g1=0	r	r	0100	RR	
RADI	g1 := g1 + g2 + 1	r<0	g1=0	r	r	0140	RR	
RSBI	g1:= g1 - g2 - 1	r<0	g1=0	r	r	0180	RR	
RI	g1 := $\overline{g2}$	g1<0	g1=0	–	–	01C0	RR	
RLD	g1 := g2	g1<0	g1=0	–	–	0200	RR	
RAD	g1 := g1 + g2	r<0	g1=0	r	r	0240	RR	
RSB	g1 := g1 – g2	r<0	g1=0	r	r	0280	RR	
RCP	form g1 - g2 and set N,Z,CA,OF	g1<g2	g1=g2	r	r	02C0	RR	
RN	g1 := g1 ∧ g2	g1<0	g1=0	–	–	0300	RR	
RO	g1 := g1 V g2	g1<0	g1=0	–	–	0340	RR	
RX	g1 := g1 ≢ g2	g1<0	g1=0	–	–	0380	RR	

MNEMONIC	EFFECT	CONDITIONS				HEX	FORMAT	CLASS
		N	Z	CA	OF			
FLD†	fa := fq	fa<0	fa=0	–	–	8000	A	F
FAD†	fa := fa + fq	fa<0	fa=0	r	–	8400	A	F
FSB†	fa : = fa - fq	fa<0	fa=0	r	–	8800	A	F
FCP†	form fa - fq and set N,Z,CA,OF	fa<fq	fa=0	–	–	8C00	A	F
FM†	ea := fa * fq	ea<0	ea=0	r	–	9400	A	F
FD†	fa := fa ÷ fq	fa<0	fa=0	r	–	9800	A	F
FST†	fq := fa	fa<0	fa=0	–	–	9C00	A	F
ELD†	ea := eq	ea<0	ea=0	–	–	A000	A	E
EAD†	ea := ea + eq	ea<0	ea=0	r	–	A400	A	E
ESB†	ea := ea - eq	ea<0	ea=0	r	–	A800	A	E
ECP†	form ea - eq and set N,Z,CA,OF	ea<eq	ea=eq	–	–	AC00	A	E
EM†	ea := ea * eq	ea<0	ea=0	r	–	B400	A	E
ED†	ea := ea ÷ eq	ea<0	ea=0	r	–	B800	A	E
EST†	eq := ea	ea<0	ea=0	–	–	BC00	A	E
FIX†	wq := integer part of ea ; ea := remainder	q <0	q=0	r	–	B000	A	W
FLT†	ea := floating representation of wq	ea<0	ea=0	–	–	9000	A	E
FNEG	ea := —ea	ea<0	ea=0	–	–	1702	L	C
SIM	Set Integer Mode (FM := 0)	–	–	–	–	1700	L	C
SFM	Set Floating Mode (FM := 1)	–	–	–	–	1701	L	C
AINT	Acknowledge Interrupt	–	–	–	–	1C00	L	C
PERM	Permit Interrupts	–	–	–	–	1C01	L	C
INH	Inhibit Interrupts	–	–	–	–	1C02	L	C
TERM	Terminate Interrupt Level	–	–	–	–	1C03	L	C
SINT	Software Interrupt	–	–	–	–	1C04	L	C
IN	Input from Channel N	S_4	S_5	S_6	S_7	1D00	L	I/O
OUT	Output to Channel N	S_4	S_5	S_6	S_7	1D40	L	I/O
SFN	Switch to Full Nucleus	–	–	–		1202	L	C
CALL	Nucleus Instructions – See CPU Nucleus Manual					1000	L	
ICBR						1100	L	
SEG						1300	L	
SEM						1400	L	

† Instructions available if FM=1 only.

Notes

(1) The class of a Format A or Format L instruction determines how its D field is used:–

FORMAT A CLASS B – D used to Form Byte Operand Address
H – D used to Form Halfword Operand Address
W – D used to Form Fullword Integer Operand Address
F – D used to Form Short Floating Point Operand Address
E – D used to Form Long Floating Point Operand Address

FORMAT L CLASS LB – D used as Literal Byte Operand
BR – D used as Signed displacement
SH – D used to define Shift operation
BIT – D used to define Bit operation
C – D used to further define operation

(2) *Condition Settings*

In the tables, the following symbols are used

All Conditions :– – indicates that the conditions bit is unaffected by the operation.

N Condition:– r<0 indicates that the bit is set to the true sign of the result.

a<0 etc. indicates that the bit is set to the sign of the result register. This may differ from the true sign if overflow has occurred.

a<hq etc. indicates that the bit is set to indicate the true result of the comparison.

CPBS only r<0 indicates that the bit is set to indicate the result of the last byte comparison performed.

Z Condition:– a=0 etc. indicates that the result register became zero as a result of the operation.

a=hq etc. indicates that the two operands were equal.

TSTB only bit = 0 indicates that the selected bit was zero.

CPBS,SCBS only end indicates that the bit is set if the operation continued to completion.

OF Condition:– r indicates that the bit is set if the result overflowed. In such cases the true result and the result placed in the result register differ.

ST only hq ≠ a indicates that the bit is set if the integer in a cannot be represented in 16 bits.

Note that once set, the OF bit can only be reset by the BOF instruction.

CA Condition:– r indicates that for Add operations, Carryout was produced, and for Subtract operations, a Borrow was produced.

SECOND HEX DIGIT

FIRST HEX DIGIT

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	FORMAT RR ②				B	+256	-512	-256	BL	+256	-512	-256				
1	CALL ③	ICB ③	KEY ④	SEG ③	SEM ③	BITX ③	BITL ③	MISC ④	PBAR	SHL ③		CPYL	INT ④	IO ③	PEC*	CPZL
2	LDL	ADL	SBL	CPL	NL	ML	DL	BS ③	BN	BNN	BZ	BNZ	BP	BNP	BOF	BNCA
3	LDXL	ADXL	SBXL	CPXL	NXL	MXL	DXL	MHS*	LDYL	ADYL	SBYL	SHX ③	SEXT*	ADZL	SBZL	LDZL

Branch

Branch and Link

FORMAT B AND L

*FOR THESE INSTRUCTIONS, THIRD AND FOURTH HEX DIGITS = 00
○FOR THESE INSTRUCTIONS, REFER TO APPROPRIATE TABLE
ALL OTHER INSTRUCTIONS – THIRD AND FOURTH HEX DIGITS DEFINE DISPLACEMENT

FUNCTION

4	LDB	LBX	DECS	CPB
5	NBS	OBS	XBS	STB
6	HAY	LDM	STM	
7	HAZ	BI	BLI	INCS
8	LD/FLD	AD/FAD	SB/FSB	CP/FCP
9	N/FIX	M/FM	D/FD	ST/FST
A	LDW/ELD	ADW/EAD	SBW/ESB	CPW/FCP
B	NW/FLT	MW/EM	DW/ED	STW/EST
C	LDX	ADX	SBX	CPX
D	NX	MX	DX	STX
E	LDY	ADY	SBY	STY
F	LDZ	DAZ	SBZ	STZ
SECOND HEX DIGIT	0 1 2 3	4 5 6 7	8 9 A B	C D E F

FORMAT A

TABLES DEFINE FUNCTION AND ADDRESSING MODE.
ADD FOURTH HEX DIGIT TO FORM DISPLACEMENT.

L = Base Reg.
I = Indirect
X = Indexed

Base Reg.

X indexed

MODE

SECOND HEX DIGIT	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0, 4, 8, C	LX	GX	YX	ZX	GI	+16	+32	+48	LIX	+16	SI	+16	YIX	+16	ZIX	+16
1, 5, 9, D	GI	+16	+32	+48	+64	+80	+96	+112	+128	+144	+160	+176	+92	+208	+224	+240
2, 6, A, E	L	+16	+32	+48	+64	+80	+96	+112	S	+16	+32	+48	+64	+80	+96	+112
3, 7, B, F	Y	+16	+32	+48	+64	+80	+96	+112	Z	+16	+32	+48	+64	+80	+96	+112

THIRD HEX DIGIT

TABLE 1

FUNCTION

0		RADC	RSBC	
1	RNA	RADI	RSBI	RI
2	RLD	RAD	RSB	RCP
3	RN	RO	RX	
SECOND HEX DIGIT	0 1 2 3	4 5 6 7	8 9 A B	C D E F

THIRD HEX DIGIT

REGISTER G1

0	A
B	X
L	S
Y	Z
0 1 2 3 4 5 6 7	8 9 A B C D E F

FOURTH HEX DIGIT

REGISTER G2

0
A
B
X
L
S
Y
Z

FORMAT RR

TABLE 2

	SHL SHX	BITL BITX	BS	CALL	SEG	SEM	ICB	I/O (Basic Test)
0	SBAR SBRX +16	TSTB TSTX	MBS *	EXIT *	LCST	REL *	ICBR *	IN
1								
2	SBAL SBLX +16			STAT	CLCS	CCLM *	ICBL *	
3								
4	SR SRX +16		CPBS *	RSEN	SCST	CLM *		OUT
5								
6	SL SRX +16			SEND	LHSR *			
7								
8	SRL SRX +16		TRBS *	RW/PIN *				
9		TGLB TGLX						
A	SLC SLCX +16	PLCB PLCX		LWCB *				
B		SETB SETX						
C			SCBS *	TRIP/OUT *				
D	SXRX SXR							
E	SXLX SXL	CLRB CLRX		LWT *				
F								

THIRD HEX DIGIT

* FOR THESE INSTRUCTIONS, FOURTH HEX DIGIT = 0.

FOR SHL/SHX, ADD 4TH HEX DIGIT TO FORM NUMBER OF PLACES OF SHIFT.

FOR BITL/BITX, 4TH HEX DIGIT IS BIT NUMBER.

FOR SEG (EXCLUDING LHSR), 4TH HEX DIGIT DEFINES SEGMENT. MUST BE IN THE RANGE 0 = 3.

FOR I/O, 4TH HEX DIGIT DEFINES CHANNEL.

FOR CALL INSTRUCTIONS STAT, RSEN, SEND, 4TH HEX DIGIT DEFINES NEXT STATE.

0	–	FREE
1	–	RUN
2	–	WAIT/PASS
3	–	WAIT
4	–	FREE CONDITIONALLY
7	–	WAIT CONDITIONALLY

TABLE 3

	INT	KEY	MISC
0	AINT	RK	SIM
1	PERM	HRK	SFM
2	INH	SFN	FNEG
3	TERM		
4	SINT		
5 – F			

FOURTH HEX DIGIT

THIRD HEX DIGIT = 0

TABLE 4