SECTION 3 - Program Instructions

List of Contents

PROGRAM INSTRUCTIONS

Introduction

There are three types of basic instructions which are grouped according to the bit format of the instruction word. These types are :-

a) Memory Reference Instructions which deal mainly with the transfer of information to/from Accumulators A and B to/from the core stores.

b) Register and Control Instructions which deal mainly with shifts, clears, rotates, negative tests, zero tests, etc. on the Accumulators.

c) Literal Instructions (I/O, Shifts, etc.) All Input/Output instructions are privileged, while the literal functions deal with multiple shifts and rotates.

The following pages describe in detail each of the instructions in the three groups. Functions of bits appearing in the form A/B, Z/C, I/D, L/R, T/F, S/R, L/S or A/L throughout these specifications are invariably obtained by coding a 1 or Ø respectively (1/Ø). Thus, for example, A is specified by a one-bit, and B by a zero-bit. The following defines the abbreviations used :-

A/B	Accumulator A/ Accumulator B
Z/C	Zero Page/Current Page
I/D	Indirect/Direct
L/R	Left/Right
T/F	True/False
S/R	Shift/Rotate
L/S	Long/Short
A/L	Arithmetic/Logical

a) MEMORY REFERENCE INSTRUCTIONS

The 22 Memory Reference instructions carry out some operation involving core locations, such as transferring information in or out of a core memory location or checking the memory location contents. Memory Reference instructions have the following general format in Machine Language, as shown in Fig.3:1 below :-

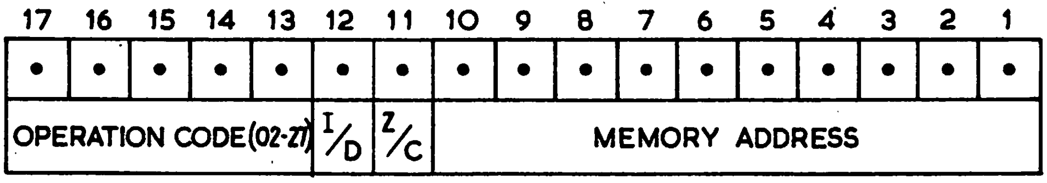


Fig. 3:1

The instruction word uses five bits (Bits 13-17) to encode the 22 instruction commands in this group. The address referenced is determined by a combination of ten memory address bits (Bits 1 to 10), plus two bits (Bits 11 and 12) to identify one of four addressing modes :-

Direct in Page Zero
Direct in Current Page
Indirect through Page Zero
Indirect through Current Page

Because there are only ten bits available to specify the memory address, a Memory reference instruction can directly reference only Octal 4000 words, 2000 on the Zero Page (the base page, consisting of locations 000000 through 001777) and 2000 on the Current Page (the page in which the instruction itself is situated). Bit 11 in a Memory Reference instruction specifies one or the other of these two pages as follows :-

1	=	Zero Page
Ø	=	Current Page

a page being defined as the largest block of memory which can be addressed by the ten memory address bits of a Memory instruction. The Molecular 18 core memory is logically divided into pages of 1024 words (decimal) each. Octal addresses of the pages are as follows :-

Page Ø	-	000000 to 001777
Page 1	-	002000 to 003777
Page 2	-	004000 to 005777
Page 3	-	006000 to 007777
Page 4	-	010000 to 011777
and so on through to :-		
Page 77	-	176000 to 177777

It will be noted that each page starts on an even thousand (Octal). Appendix 15 lists the octal addresses of all pages from Ø to 77 in full.

To address locations in any page other than Zero or Current, indirect addressing is used via the Zero or Current Page, as explained beneath :-

Indirect/Direct Addressing

Memory Reference instructions include a bit (Bit 12) reserved to specify direct or indirect addressing. Direct addressing combines the operation code and the effective address into one word, permitting a Memory Reference instruction to be executed in two machine cycles (Fetch and Execute), except for Jump, which takes only one machine cycle. Indirect addressing uses the address part of the instruction to access another word in memory which is taken as a new memory reference for the same instruction. This new address is a full 17 bits long, 16 bits of address plus another bit (Bit 17) which is used as a further Indirect/Direct bit. The 16-bit length of the address permits access to any location in up to 64K of core. If Bit 17 again specified indirect addressing still another address is obtained; this multiple-step indirect addressing could be carried out to any number of levels (up to 15), but this of course would not be practical. The first address obtained in the Indirect phase which does not specify another indirect level becomes the effective address for the Memory instruction. Memory instructions with indirect addresses are therefore executed in a minimum of three machine cycles (Fetch, Indirect and Execute).

Indirect or Direct addressing is specified by Bit 12 (or Bit 17) as follows :-

1	=	Indirect
Ø	=	Direct

Note that since Accumulators A and B can be addressed, a Memory Reference instruction can apply to either of these registers, both directly or indirectly (except in the case of a <u>direct</u> JUMP or JSBR), as well as to core stores.

Fig. 3:2 gives instruction codes and mnemonics for all Memory Reference instructions :-

		17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Mnemonic *	Octal	INSTRUCTION					I/D	Z/C	MEMORY ADDRESS									
JUMP △	02	0	0	0	1	0												
JSBR △	03	0	0	0	1	1												
INSZ △	04	0	0	1	0	0												
DESZ △	05	0	0	1	0	1												
ANDA △	06	0	0	1	1	0												
IORA △	07	0	0	1	1	1												
XORA △	10	0	1	0	0	0												
ADA △	11	0	1	0	0	1												
ADB △	12	0	1	0	1	0												
SFA △	13	0	1	0	1	1												
SFB △	14	0	1	1	0	0												
ADAC △	15	0	1	1	0	1												
ADBC △	16	0	1	1	1	0												
SFAC △	17	0	1	1	1	1												
SFBC △	20	1	0	0	0	0												
LDA △	21	1	0	0	0	1												
LDB △	22	1	0	0	1	0												
CMPA △	23	1	0	0	1	1												
CMPB △	24	1	0	1	0	0												
STA △	25	1	0	1	0	1												
STB △	26	1	0	1	1	0												
UNSTK △	27	1	0	1	1	1												

Fig. 3:2 Memory Reference Instructions

* The Mnemonic code is meaningful to and translated by the Assembler into binary code. In Usercode, Indirect is specified by attaching I to the function mnemonic before the space, which is mandatory.

There now follows an explanation of each instruction in the Memory Reference Group. In each individual instruction description, the name will be given first, followed by the appropriate assembler mnemonic.

Unconditional Jump - JUMP

Unconditional JUMP to the word of core specified by the operand. The JUMP instruction loads the effective address of the instruction into the Program Counter (PC), thereby changing the program sequence, since the PC specifies the next instruction to be performed. The next instruction is then taken from that location, the program continuing operation from there. The JUMP instruction does not, in any way, affect the contents of the Accumulators. It should be noted that a JUMP Direct to an Accumulator will give unpredictable results.

Jump to Subroutine - JSBR

On this instruction the program will Jump to the word of core specified by the operand. During execution of the instruction the current PC is incremented by one and stored in Accumulator B, replacing or over-writing any original contents. After the subroutine is executed, this pointer address (of the JSBR instruction + 1), in other words the return or link address, identifies the next instruction to be executed. Thus, the programmer has at his or her disposal a simple means of exiting from the normal flow of the program to perform an intermediate task, and a means of return to the correct location on completion of that task, perhaps ending with a Jump Indirect via Accumulator B. It should be noted that a JSBR Direct to an Accumulator will give unpredictable results.

Increment, and Skip if Zero - INSZ△

This instruction adds one to the contents of the word of core specified (the operand), all 17 bits, and then examines the result of the addition.

If the result is zero, the instruction following the INSZ is skipped and the Carry flag will be set.

If the result is not zero, the program will proceed normally to the instruction immediately following the INSZ (the next word of program in sequence).

The following points should be kept in mind with reference to the INSZ instruction :-

1) The contents of Accumulators A and B are not disturbed unless the instruction specifies A or B as the operand.

2) The original word in the referenced memory location is replaced by the incremented value.

3) The INSZ instruction performs the incrementation first and then checks for a zero result.

4) The Carry flag is set whenever a transfer occurs between Bits 16 and 17 in any store as a result of the INSZ instruction.

Decrement, and Skip if Zero - DESZ△

This instruction subtracts one from the contents of the word of core specified (the operand), all 17 bits, and then examines the result of the subtraction.

If the result is zero, the instruction following the DESZ is skipped.

If the result is not zero, the program will proceed normally to the instruction immediately following the DESZ (the next word of program in sequence).

Should the specified store overflow as a result of this instruction the Carry flag will be set.

The following points should be kept in mind with reference to the DESZ instruction :-

1) The contents of Accumulators A and B are not disturbed unless the instruction specifies A or B as the operand.

2) The original word in the referenced memory location (in other words the operand), is replaced by the decremented value.

3) The DESZ instruction performs the decrementation first and then checks for a zero result.

4) The Carry flag is set whenever a transfer occurs between Bits 17 and 16 in any store as a result of the DESZ instruction. (e.g. If the operand contains Bit 17 only and is decremented, it will afterwards contain Bits 16 to 1 inclusive, and the Carry flag will be set).

"And" to A - ANDAA

The ANDA instruction causes a bit-by-bit Boolean AND operation between the contents of Accumulator A and the contents of the operand specified. The result is left in Accumulator A, replacing its original contents, but the operand specified is not altered.

Figure 3:3 below, showing a simple circuit with two switches, explains the AND operation. Should current be allowed to flow through a switch, the switch is said to have a value of '1'. Where current cannot flow through because the switch is open, the switch is said to have a value of 'Ø'. When the whole circuit is considered it will be noted that current may only flow through it (therefore giving it a value of '1'), when both switches are '1'. This is the AND operation.

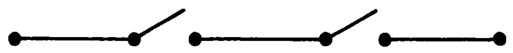


Fig. 3:3

When this is applied to binary numbers, a binary 1 will result if a binary 1 appears in the relevant position of the two numbers, but a Ø will result if only one of the bits has this value. Accordingly, the ANDA instruction can be used to Mask Out a portion of a number or word.

e.g.	To be Masked Out	retained for future use	
	10111010101	010101	(17-bit word in Accumulator A)
	00000000000	111111	(Mask in operand)
	00000000000	010101	(Result left in Accumulator A)

The following points sum up the ANDA instruction :-

1) A '1' is left in Accumulator A only when a '1' is present in the corresponding position of both Accumulator A and the specified word of core (Mask).

2) The Carry flag is not affected, neither is the Greater Than flag, as the operation is performed on a bit-for-bit basis.

3) The operand specified remains unaltered.

Acc. A	Operand	Result in A
O	O	O
O	I	O
I	O	O
I	I	I

Fig. 3:4

"Inclusive OR' to A - IORAA

The IORA instruction causes a bit-by-bit Boolean OR (or Inclusive OR) operation between the contents of Accumulator A and the contents of the operand specified by the IORA instruction. The result is left in A, replacing its original contents, but the operand is not altered.

The circuit diagram in Figure 3:5 illustrates the use of the OR connective when two words are combined. It can be seen that current is allowed to flow to Z whenever EITHER X or Y switches (or both) is closed.

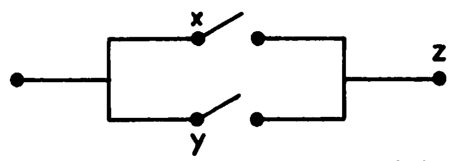


Fig. 3:5

i.e. Z = 1 if X = 1 or Y = 1
Z = 1 if X = 1 and Y = 1

Correspondingly, the IORA instruction results in the value of 1 if either or both relevant bits of Accumulator A and the operand is 1.

e.g.		
	A contents	110110
	Operand	011010
	Result (in A)	111110

The following points sum up the IORA instruction :-

1) A '1' is inserted in Accumulator A if a 1 is present in the corresponding position of <u>either</u> Accumulator A <u>or</u> the operand.

2) The Carry flag is not affected, neither is the Greater Than flag.

3) The specified word of core (operand) remains unchanged.

Acc. A	Operand	Result in A
0	0	0
0	1	1
1	0	1
1	1	1

Fig. 3:6

"Exclusive OR" to A - XORA∆

The XORA instruction causes a bit-by-bit Boolean "Exclusive OR' operation between the contents of Accumulator A and the contents of the operand specified by the XORA instruction. The result is left in Accumulator A, replacing its original contents, but the operand specified is not altered.

The circuit diagram in Fig. 3:7 below illustrates that the Exclusive OR is similar to the Inclusive OR with the exception that one set of conditions has been altered or excluded. This exclusion has been shown by connecting the two switches mechanically together so that it is impossible for them to be closed simultaneously, although they may be open simultaneously or singly.

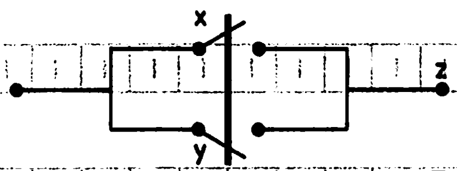


Fig. 3:7

i.e. Z = 1 if X = 1 and Y = ∅
Z = 1 if X = ∅ and Y = 1

Correspondingly, the XORA results in the value of 1 if only one of the relevant bits of Accumulator A and the operand is 1.

e.g.

```
        A Contents      110110
        operand         011010
        Result (in A)   101100
```

The following points sum up the XORA instruction :-

1) A '1' is inserted in Accumulator A only if the two corresponding bits of the Accumulator and the operand differ.

2) The Carry flag is not affected, neither is the Greater Than flag.

3) The operand remains unaltered.

Acc. A	Operand	Result in A
O	O	O
O	I	I
I	O	I
I	I	O

Fig. 3:8

Add to A - ADA△

The ADA instruction performs a binary addition between the specified word and the contents of Accumulator A, all 17 bits, leaving the result of the addition in Accumulator A. The specified word of core will remain unchanged, but the result of the addition could set the Carry flag. The following figure (Fig. 3:9) illustrates the operation of the ADA instruction :-

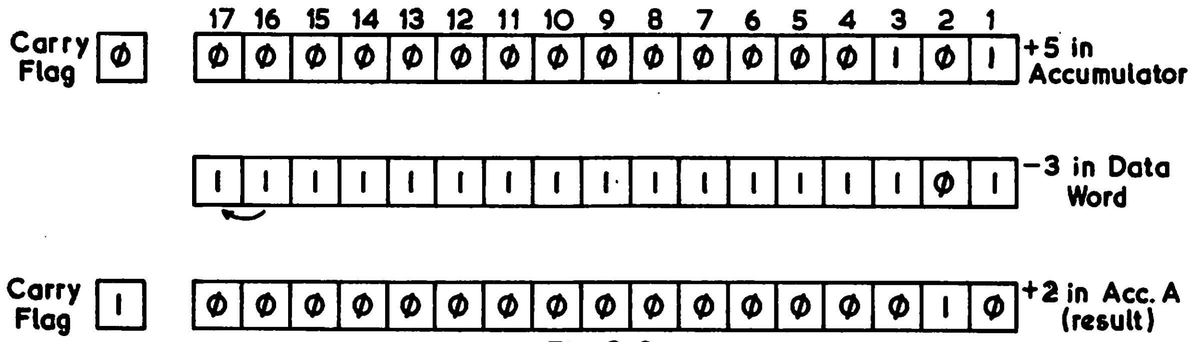


Fig. 3:9

In the example above, a carry has occurred between Bits 16 and 17, and this has caused the Carry flag to be set. This could be detected with a Register Instruction 'Skip if Carry'. Arithmetically the sign bit (Bit 17) behaves in exactly the same way as the other 16 bits. As can be seen above, the most significant bit of the answer was lost off the end of the register, as would also happen if two stores containing Bit 17 only were added together.

Add to B - ADB△

The ADB instruction performs a binary addition between the specified word and the contents of Accumulator B, all 17 bits, leaving the result of the addition in Accumulator B. The specified word of core will remain unchanged, but the result of the addition could set the Carry flag.

Subtract from A - SFA△

The SFA instruction performs a binary subtraction, subtracting the contents of the specified word of core from the contents of Accumulator A, all 17 bits, leaving the difference in Accumulator A. The specified word of core will remain unchanged, but the result of the subtraction could set the Carry flag.

Arithmetically, binary numbers may be directly subtracted in a manner similar to decimal subtraction. The essential difference is that if a 'borrow' is required, it is equal to the base of the system of 2.

```
i.e.                    110-   =   6    (decimal)
                        101    =   5
                        ───        ─
                        001        1
```

To subtract 1 from Ø in the first column, a borrow of 1 was made from the second column, which effectively added 2 to the first column. After the borrow, 2 - 1 = 1 in the first column; in the second column Ø - Ø = Ø; and in the third column 1 - 1 = Ø.

The following Figure 3:10 illustrates the operation of the SFA instruction :-

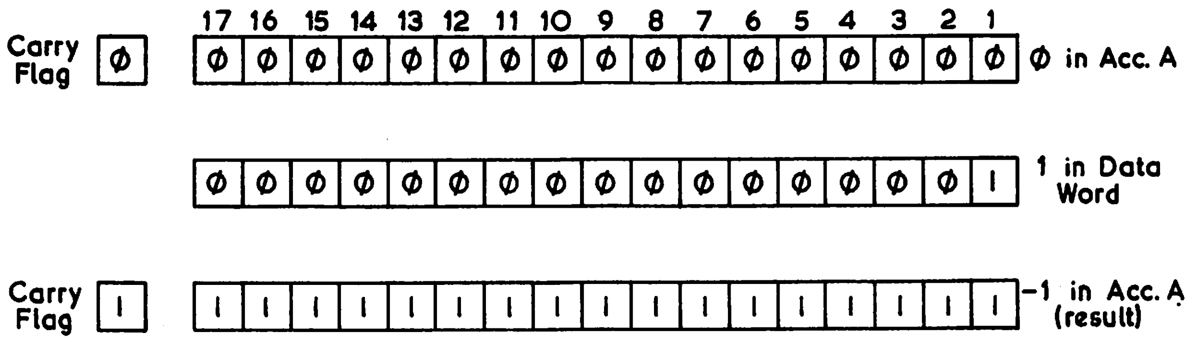


Fig. 3:10

In the above example the 'borrow' is lost off the end of the register; also a carry has occurred between Bits 16 and 17 which has caused the Carry flag to be set. Again this could be detected with the register instruction 'Skip if Carry'.

Subtract from B - SFBΔ

The SFB instruction performs a binary subtraction, subtracting the contents of the specified word of core from the contents of Accumulator B, all 17 bits, leaving the difference in Accumulator B. The specified word of core remains unchanged, but the result of the subtraction could set the Carry flag.

Add with Carry (to Accumulator A) - ADACA

Multiple register addition is possible using the ADAC instruction. This type of arithmetic is needed when a number that is too large to be contained in one word (i.e. more than 65,535 in decimal), has to be added to another similar number, or when the result of an addition may be too large for a single register.

e.g. Two numbers 65,535 and 65,537 are to be added together

```
65,535  =  00000 0000 0000 0000 01111 1111 1111 1111
65,537  =  00000 0000 0000 0001 00000 0000 0000 0001
```

Let us assume that the 65,535 is double stored in Stores 0235 and 0236, the 65,537 is double stored in Stores 0251 and 0252, and that the answer is to be stored in 0276 and 0277. The first bit of program could read :-

```
CLC          (Clear Carry)
LDA   0236   (Load A with 0236)    01111 1111 1111 1111
ADA   0252   (Add to A 0252)       00000 0000 0000 0001
                                   ____________________

             (Result in A)         10000 0000 0000 0000
```

As Bit 17 is left on the presence of this bit denotes a negative number, therefore it must be cleared before being stored in the answer store :-

```
CLSA         (Clear Sign of A)
STA   0277   (Store A in 0277)     00000 0000 0000 0000
```

The first register has been added, and because a carry occurred between Bits 16 and 17 the Carry flag will be set. Using ADAC for the addition of the second pair of stores, the "with carry" will cause the Carry flag to be added to the A register before the addition is started. The addition is then completed normally.

```
LDA   0235         00000 0000 0000 0000
                                      1  - Carry flag
ADAC  0251         00000 0000 0000 0001
                   ____________________

STA   0276         00000 0000 0000 0010
```

The total left in Stores 0276 and 0277 now equals :-

```
00000 0000 0000 0010 00000 0000 0000 0000  =  131,072 (Decimal)
```

To sum up, on an ADAC instruction, the contents of the Carry flag (if any) will be added to the contents of Accumulator A, the Carry flag will be cleared, and the contents of the word of core specified will then be added to Accumulator A. The store specified will remain unchanged.

N.B. It should be pointed out that the Carry flag could be set again by the addition.

This instruction is used for double or multiple store arithmetic.

Add with Carry (to Accumulator B) - ADBCΔ

Exactly as for ADAC, except that the contents of the Carry flag (if any) will be added to the contents of Accumulator B, the Carry flag will be cleared, and the contents of the word of core specified will then be added to Accumulator B. The store specified will remain unchanged.
N.B. The Carry flag could be set again by the actual addition.

This instruction is used for double or multiple store arithmetic.

Subtract Store from A with Carry - SFACΔ

The contents of the Carry flag (if any) will be subtracted from the contents of Accumulator A, the Carry flag will be cleared, and the contents of the word of core specified will then be subtracted from Accumulator A. The store specified will remain unchanged.
N.B. The Carry flag could be set again by the actual subtraction.

This instruction is used for double or multiple store arithmetic.

Subtract store from B with Carry - SFBCΔ

The contents of the Carry flag (if any) will be subtracted from the contents of Accumulator B, the Carry flag will be cleared, and the contents of the word of core specified will then be subtracted from Accumulator B. The store specified will remain unchanged.
N.B. The Carry flag could be set again by the actual subtraction.

This instruction is used for double or multiple store arithmetic.

Load into A - LDAΔ

LDA stores the contents of the referenced location in Accumulator A, over-writing the original contents of Accumulator A. The specified word of core remains unaltered.

Load into B - LDBΔ

LDB stores the contents of the referenced location in Accumulator B, over-writing the original contents of Accumulator B. The specified word of core remains unaltered.

Compare store with A (skip if unequal) - CMPA△

This instruction compares the contents of the word of core specified with the contents of Accumulator A, all 17 bits. If the two words are different the next instruction will be skipped (i.e. the PC is advanced by two instead of one). If both words are identical, the program will proceed normally to the next instruction in sequence. The contents of both the specified word of core and Accumulator A remain unaltered.

Should the contents of Accumulator A be greater than the contents of the word of core addressed, the Greater Than flag will be set (but NOT the Carry flag).

Should the contents of Accumulator A be less than or equal to the contents of the word of core addressed, the Greater Than flag will be cleared (but NOT the carry flag).

Compare store with B (skip if unequal) - CMPB△

This instruction compares the contents of the word of core specified with the contents of Accumulator B, all 17 bits. If the two words are different the next instruction will be skipped (i.e. the PC is advanced by two instead of one). If both words are identical, the program will proceed normally to the next instruction in sequence. The contents of both the specified word of core and Accumulator B remain unaltered.

Should the contents of Accumulator B be greater than the contents of the word of core addressed, the Greater Than flag will be set (but NOT the Carry flag).

Should the contents of Accumulator B be less than or equal to the contents of the word of core addressed, the Greater Than flag will be cleared (but NOT the Carry flag).

Store A - STA△

The STA instruction stores the contents of Accumulator A in the word of core specified, over-writing the original contents of the referenced location. Accumulator A remains unaltered.

Store B - STB△

The STB instruction stores the contents of Accumulator B in the word of core specified, over-writing the original contents of the referenced location. Accumulator B remains unaltered.

Unstack - UNSTKΔ

This is a Privileged Instruction, and may be used only in the system software. It has been assigned the function "Unstack", the address specified normally referring to 000030, thus pointing to that part of the Interrupt Stack indicated by the Interrupt Stack Pointer.

The 'Unstack' instruction first restores the contents of Accumulators A and B, the Base Register, the Program Counter and the Carry and Greater Than flags and then :-

(i) Uses the sign bit of the 2nd word in the appropriate stack to set the Addressing Mode Memory (see Page 1:6) to the correct state.

(ii) Reduces the Interrupt Stack Pointer by 5.

(iii) Uses Step 01 relative to the Base to set the Limit Register.

(iv) Sets Interrupt on.

(v) Causes a JUMP to the step specified in the PC.

b) REGISTER AND CONTROL INSTRUCTIONS

These instructions, in general, manipulate bits in the A and B accumulators. There is no reference to memory, thus the instructions are executed in only one machine cycle. Register and Control Instructions have the following general format in Machine Language :-

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
OPERATION CODE (00)					MODE		A/B	REGISTER MICRO – INSTRUCTIONS								

Fig. 3:11

The instruction word uses Bits 17-13 to specify the type of instruction (being all zeros for Register and Control instructions), Bits 12 and 11 to specify the Mode, Bit 10 to specify the relevant Accumulator and Bits 9 to 1 to combine in 'micro-instructions', with the resulting multiple instruction operating on the A or B accumulators as a single-word instruction.

Micro-instructions may be combined under the following general rules :-

1) No instructions may be used which combine micro-instructions from different Modes.

2) References to both A and B registers cannot be mixed in the same micro-instruction.

3) If more than one of the micro-instruction bits of a particular Mode is set at any one time, the sequence of execution is left to right (Bit 9 to Bit 1).

4) If two (or more) skip functions are combined, the skip will only occur if all conditions are fulfilled. One exception exists: In Mode 1 only, the skip will occur if either or both conditions are met.

5) Shift and Rotate or Increment and Decrement must not be mixed in the same micro-instruction - the effect of doing so is undefined.

MODE - there are four modes - the mode alters the meaning of the micro-instruction bits.

Mode 0	Mode 2
Mode 1	Mode 3

Mode 0

The Mode 0 instructions contain 27 basic instructions, only 7 of which will be used by the Applications programmer, the other 'control' instructions being 'Privileged', and only used by the system software. Each Mode 0 instruction has its own binary structure as shown in Fig. 3:12 below. These instructions are not micro-programmable.

	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Octal Representation
Mnemonics	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
NOP;	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	000000
HALT;	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	000001
MASK;	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	000002
ACK;	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	000003
ION;	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	000004
IOF;	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	000005
SKON;	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	000006
SKOF;	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	000007
SKMF;	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	000010
SKMR;	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	000011
SKMP;	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	000012
SKPM;	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	000014
SKSW;	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	000015
SKIC;	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	000016
RIO;	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	000017
SK7L;	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	000020
SK15;	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	000021
SKTI;	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	000022
SKIF;	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	000023
SKEX;	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	000024
SRLD;	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	000540
SRAD;	0	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	000560
SLLD;	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	000740
SLAD;	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	000760
EXC;	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	001000
SETGT;	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	001001

Fig. 3:12

The operation of the 7 Mode 0 instructions in general use is first described below :-

No Operation - NOP;

If all bits are zero, no operation is performed and program control is transferred to the next instruction in sequence.

Shift Right Logical Double Length (one place) - SRLD;

This instruction causes a double length single shift, in that a double length Accumulator (comprised of Accumulators A and B) is shifted right one place, as per Fig. 3:13 below :-

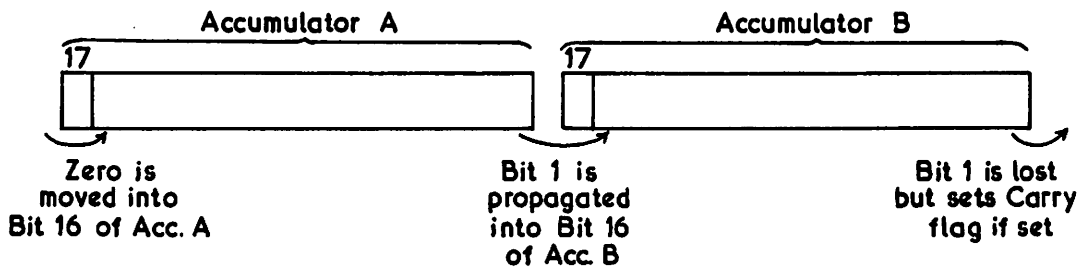

Accumulator A
Accumulator B
17
17
Zero is moved into Bit 16 of Acc. A
Bit 1 is propagated into Bit 16 of Acc. B
Bit 1 is lost but sets Carry flag if set

Fig. 3:13

Bits 16 to 1 of both accumulators are shifted one place to the right, Bit 17 (the sign bit) of each remaining stationary. A zero is moved into position 16 of Accumulator A. The contents of Bit 1 of Accumulator A are propagated into Bit 16 of Accumulator B, but without affecting the Carry flag. Should there be a '1' bit in position 1 of Accumulator B the Carry flag will be set, the '1' bit being otherwise lost. (If there is a 'Ø' bit in position 1 of Accumulator B the Carry flag, should it be set already, will not be overwritten.)

Shift Right Arithmetic Double length (one place) - SRAD;

This instruction causes a double length single shift, in that a double length accumulator (comprised of Accumulators A and B) is shifted right one place, as per Fig. 3:14 below :-

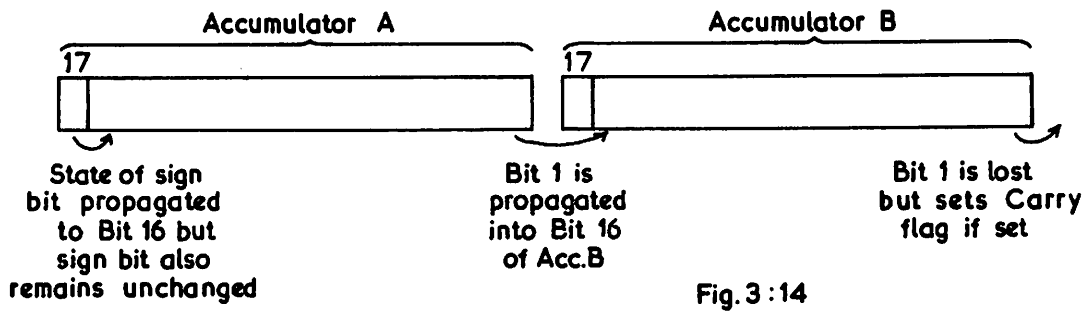

Accumulator A
Accumulator B
17
17
State of sign bit propagated to Bit 16 but sign bit also remains unchanged
Bit 1 is propagated into Bit 16 of Acc.B
Bit 1 is lost but sets Carry flag if set

Fig. 3:14

Bits 16 to 1 of both accumulators are shifted one place to the right, Bit 17 (the sign bit) of each remaining unchanged, although the state of the sign bit of Accumulator A is propagated into Bit 16 of Accumulator A. The contents of Bit 1 of Accumulator A are propagated into Bit 16 of Accumulator B, but without affecting the Carry flag. Should there be a '1' bit in position 1 of Accumulator B, the Carry flag will be set, the '1' bit being otherwise lost. (If there is a 'Ø' bit in position 1 of Accumulator B, the Carry flag, should it be set already, will not be overwritten.)

Shift Left Logical Double length (one place) - SLLD;

This instruction causes a double length single shift, in that a double length Accumulator (comprised of Accumulators A and B) is shifted left one place as per Fig. 3:15 below :-

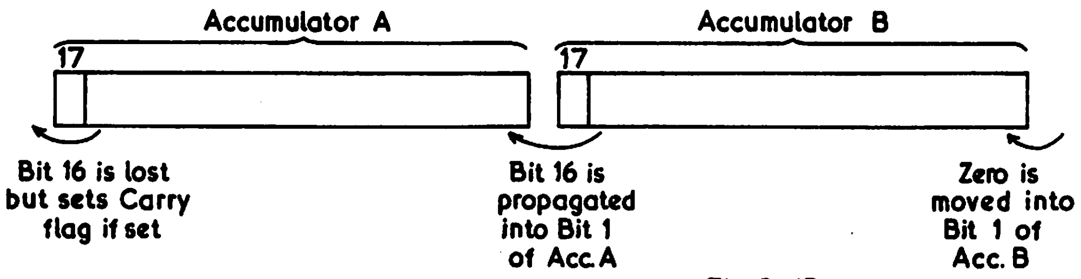

Accumulator A
Accumulator B
17
17
Bit 16 is lost but sets Carry flag if set
Bit 16 is propagated into Bit 1 of Acc.A
Zero is moved into Bit 1 of Acc. B

Fig. 3:15

Bits 16 to 1 of both accumulators are shifted one place to the left, Bit 17 (the sign bit) of each remaining stationary. A zero is moved into position 1 of Accumulator B. The contents of Bit 16 of Accumulator B are propagated into Bit 1 of Accumulator A, but without affecting the Carry flag. Should there be a '1' bit in position 16 of Accumulator A, the Carry flag will be set, the '1' bit being otherwise lost. (If there is a 'Ø' bit in position 16 of Accumulator A the Carry flag, should it be set already, will not be overwritten.)

Shift Left Arithmetic Double length (one place) - SLAD;

This instruction is identical to SLLD (above), as per Fig. 3:16 below :-

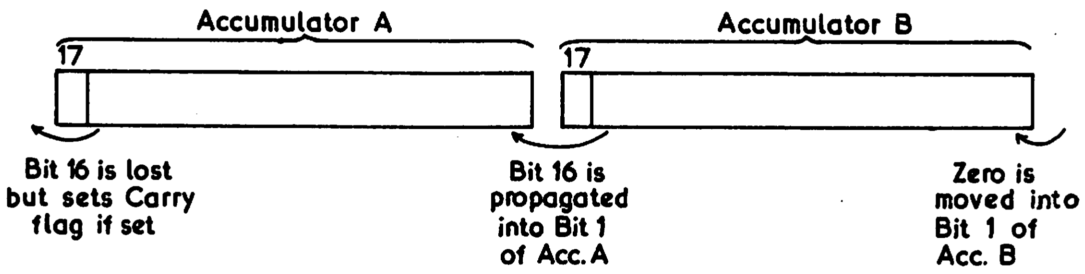

Accumulator A
Accumulator B
17
17
Bit 16 is lost but sets Carry flag if set
Bit 16 is propagated into Bit 1 of Acc.A
Zero is moved into Bit 1 of Acc. B

Fig. 3:16

User call of Executive - EXC;

This instruction is used when Executive is needed to carry out a task. It gives rise to an internal interrupt, which eventually causes the program to go to location 000210.

Set Greater Than Flag - SETGT;

This instruction causes the Greater Than flag to be set.

The operation of each individual 'Privileged' control instruction is briefly described below :-

<u>Halt - HALT;</u>

If Bit 1 is set and all other bits are zero, the computer will stop at the conclusion of the current machine cycle.

<u>Mask Out - MASK;</u>

This instruction sets up the Interrupt Disable flags of each device, according to a pattern or mask set up in Accumulator A - each device's Interrupt Disable flag is set or cleared as the corresponding bit in the Mask is 1 or Ø. (The Mask Bit No. chart is shown in Fig. 3:17 below.)

<u>Mask Bit Numbers</u>

Device	Mask Bit No.
Tape Reader	1
Card Reader	1
Single Shot	1
Line Printer	2
Serial Printer	2
Display (Alpha/Numeric)	2
Tally Roll Printer	2
Keyboards	3
Card Feeds	4
Form Feeds	5
Punch	6
80 Column Card Reader	6
IBM Input	7
IBM Output	8
Modem Coupler transmit	9
Modem Coupler receive	10
Disc	16

Fig.3:17

The Mask Bit numbers refer to I/O Bus Bits 1 to 17 numeric with ascending Binary order. Every device is wired to a particular data line on the in-out bus and hence to a particular bit of the mask. Although slower devices are assigned to the higher numbered bits in the mask, there is no established priority as the program can use any mask configuration.

Acknowledge Interrupt - ACK;

This instruction determines which is the highest priority device awaiting service by reading its device code into Accumulator B, assuming an I/O Interrupt is pending. ACK can read the code of only one device at a time, whichever of those waiting has the highest priority. This is normally determined by the positions of the I/O Boards in the chassis, i.e. the further from the processor, the lower the priority. When two devices share a single I/O Board (e.g. Keyboard and Fast Serial Printer/ Display) the relative priority is fixed (Keyboard higher).

Interrupt On - ION;

This instruction sets the Interrupt On flag, to allow the processor to respond to interrupt requests. If interrupt is disabled when this instruction is given, the CPU executes the next instruction (step) and then enables interrupt.

Interrupt Off - IOF;

This instruction clears the Interrupt On flag to prevent the processor from responding to I/O interrupt requests, and also prevents all internal processor interrupts.

Skip if Interrupt On - SKON;

This will skip the next instruction if interrupt is on.

Skip if Interrupt Off - SKOF;

This will skip the next instruction if interrupt is off.

Skip if Mains Failure Interrupt - SKMF;

This will skip the next instruction in sequence when an internal interrupt is called by a power failure. If the skip is taken, the Interrupt will be reset.

Skip if Mains Return Interrupt - SKMR;

This will skip the next instruction in sequence when an internal interrupt is called when power is restored. If the Skip is taken, the Interrupt will be reset.

Skip if Memory Parity Interrupt - SKMP;

This will skip the next instruction in sequence when an internal interrupt is called in the case of a memory parity failure. If the skip is taken, the interrupt and also the lamp will be reset.

Skip if Memory Boundary Interrupt (Limit) - SKPM;

This will skip the next instruction in sequence when an internal interrupt is called by the program selecting an address outside the bounds of the Limit Register. If the skip is taken, the interrupt will be reset. (The Memory Boundary Interrupt is only applicable when in User mode.)

Skip if MA = Switch Register - SKSW;

This will skip the next instruction in sequence when an internal interrupt is called when the memory address equals an address previously set on the Data switches. If the skip is taken the interrupt will be reset.

Skip if Continuous Interrupt Switch Interrupt - SKIC;

This will skip the next instruction in sequence where an internal interrupt is called after each step. If the skip is taken, the interrupt will be reset.

I/O Reset - RIO;

This instruction clears the flags of all Input/Output devices connected to the computer.

Skip if 7th Level Interrupt - SK7L;

This will skip the next instruction in sequence when an internal interrupt is called if a certain one-bit memory register has been set by entry of the 7th level stack. If the skip is taken, the interrupt will be reset.

Skip if Greater Than 15 Indirects Interrupt - SK15;

This will skip the next instruction in sequence when an internal interrupt is called by more than fifteen indirects being 'chained'. If the skip is taken, the interrupt will be reset.

Skip if Timer Interrupt - SKTI;

Skip if Illegal Function Interrupt - SKIF;

This will skip the next instruction in sequence when an internal interrupt is called by a 'Privileged' instruction being used by program while the Addressing Mode Memory is set to zero. These instructions cannot be allowed to be used by a User program running under the Operating System, as they are likely to interfere with the running of other programs. For the purposes of this specification, an 'Illegal Function' is defined as :-

a) All Input/Output Instructions without exception

b) All Register and Control Instructions in Mode 0 with the exception of the 7 instructions mentioned on Page 3:17.

If the skip is taken, the interrupt will be reset.

Skip if Extracode Interrupt - SKEX;

This will skip the next instruction in sequence when an interrupt is called by the 'EXC' instruction (see page 3:18). If the skip is taken, the interrupt will be reset.

Mode 1

The Mode 1 instructions include 21 basic instructions which can manipulate the contents of Accumulators A or B and the Carry flag. These instructions are micro-programmable; that is, they can be combined with other instructions also in Mode 1 to perform specialised operations (see also Page 3:24). There now follows a selection table for Mode 1.

Mnemonics	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Octal Representation
	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
CLC;	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	002400
LSA;	0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	0	003300
LSB;	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	002300
RSA;	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	003100
RSB;	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	002100
LRA;	0	0	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	003240
RRA;	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	003040
LRAC;	0	0	0	0	0	0	1	1	0	1	0	1	1	0	0	0	0	003260
RRAC;	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0	003060
LRB;	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	002240
RRB;	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	002040
LRBC;	0	0	0	0	0	0	1	0	0	1	0	1	1	0	0	0	0	002260
RRBC;	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	002060
DECA;	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	003010
DECB;	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	002010
INCA;	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	003004
INCB;	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	002004
AMSB;	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	003002
BMSB;	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	002002
ALSB;	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	003001
BLSB;	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	002001

Fig.3:18

Micro-programming

If, for instance, it is desired to clear Carry and Left Shift Accumulator A, to perform this task the program could include the following instructions (given in both mnemonic and octal form).

```
CLC;   002400
LSA;   003300
```

However, when the Mode 1 instruction format is analysed, the following can be seen :-

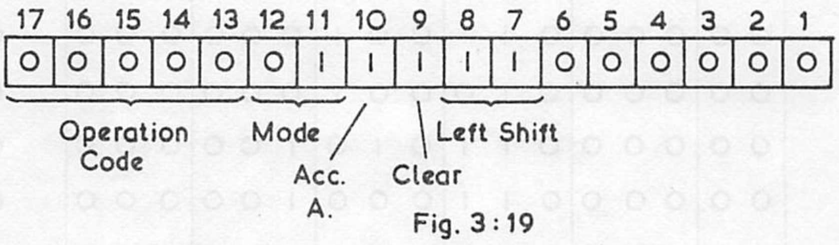


Fig. 3:19

Since the CLC and LSA instructions occupy separate bit positions they may be used in the same instruction, thus combining the two operations into one instruction. This instruction would be written in Usercode as CLC,LSA; which is 003700 in octal. In this manner, many more Register instructions, separated by a comma, may be combined to make the execution of the program more efficient.

The operation of each individual instruction specified by Bits 10 to 1 is described below :-

Clear Carry - CLC;

This instruction causes the Carry flag to be cleared.

Left Shift A or B - LSA; or LSB;

Bits 16 to 1 of the specified accumulator will be shifted one place to the left. Bit 17 (the sign bit) of the accumulator to be shifted remains stationary.

If there is a '1' bit in position 16 of the accumulator, the Carry flag will be set after a left shift.

If there is a '0' bit in position 16 of the accumulator, the Carry flag, should it be set already, will not be overwritten after a Left Shift.

Right Shift A or B - RSA; or RSB;

Bits 16 to 1 of the specified accumulator will be shifted one place to the right. Bit 17 of the accumulator to be shifted remains stationary.

If there is a '1' bit in position 1 of the accumulator, the Carry flag will be set after a right shift.

If there is a '0' bit in position 1 of the accumulator, the Carry flag, should it be set already, will not be overwritten after a Right Shift.

Left Rotate A or B - LRA; or LRB;

This instruction rotates Bits 1 to 16 of the specified accumulator one place to the left.

i.e. One left rotate will perform as for one left shift, except that the figure which was in Bit 16 will re-enter the Accumulator at Bit 1. The Carry flag is not affected. In other words, it treats Bits 1 to 16 of the specified accumulator as a closed loop, and performs what is commonly called a circular shift, meaning that any bit rotated off the left end will re-appear at the right end.

```
e.g.      one left rotate of  0110  1100  0110  0101
                       gives  1101  1000  1100  1010
```

Right Rotate A or B - RRA; or RRB;

This instruction rotates Bits 1 to 16 of the specified accumulator one place to the right.

i.e. One right rotate will perform as for one right shift, except that the figure which was in Bit 1 will re-enter the Accumulator at Bit 16. The Carry flag is not affected. In other words, it treats Bits 1 to 16 of the specified accumulator as a closed loop, and performs what is commonly called a circular shift, meaning that any bit rotated off the right end will re-appear at the left end.

```
e.g.      one right rotate of  0110  1100  0110  0101
                        gives  1011  0110  0011  0010
```

Left Rotate A or B with Carry - LRAC; or LRBC;

This instruction causes any previous Carry flag to be introduced into the gap left by the rotate, and the bit rotated off the left end will replace the Carry flag. In other words, it treats Bits 1 to 16 plus the Carry flag as a closed loop, and performs a circular shift as previously described.

```
e.g.              0110  1100  0110  0100  Carry Flag is 1

          after one left rotate with Carry would read :-

                  1101  1000  1100  1001  Carry Flag is Ø
```

Rotate with Carry is used for multiple store shifting.

Right Rotate A or B with Carry - RRAC; or RRBC;

This instruction causes any previous Carry flag to be introduced into the gap left by the rotate, and the bit rotated off the right end will replace the Carry flag. In other words it treats Bits 1 to 16 plus the Carry flag as a closed loop, and performs a circular shift as previously described.

e.g. 0110 1100 0110 0100 Carry flag is 1

after one right rotate with Carry would read :-

1011 0110 0011 0010 Carry Flag is Ø

Rotate with Carry is used for multiple store Shifting.

Decrement A or B - DECA; or DECB;

This instruction will decrement the contents of the specified Accumulator by 1.

Increment A or B - INCA; or INCB;

This instruction will increment the contents of the specified accumulator by 1.

Skip if Bit 16 of A or B equals zero - AMSB; or BMSB;

This instruction causes the program to skip the next step if Bit 16 of the specified accumulator is zero.

Skip if Bit 1 of A or B equals zero - ALSB; or BLSB;

This instruction causes the program to skip the next step if Bit 1 of the specified accumulator is zero, or, in other words, if there is an even number in the accumulator.

Mode 2

This Mode includes 15 basic instructions which can clear and complement the contents of Accumulators A and B and the Carry flag. These instructions are micro-programmable, as previously explained. There now follows a selection Table for Mode 2 :-

Mnemonics	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Octal Representation
	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
CLA;	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	005400
CLB;	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	004400
COMPA;	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	005200
COMPB;	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	004200
CLC;	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	004100
COMPC;	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	004040
SKIP;	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	004020
SWAPA;	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0	005010
SWAPB;	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	004010
CLSA;	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	005004
CLSB;	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	004004
COMPSA;	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	005002
COMPSB;	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	004002
ESWRA;	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	005001
ESWRB;	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	004001

Fig. 3:20

The operation of each individual instruction specified by Bits 10 to 1 is described below :-

Clear A or B - CLA; or CLB;

This instruction sets the specified accumulator (all 17 bits) to zeros.

Complement A or B - COMPA; or COMPB;

This instruction causes the specified accumulator (all 17 bits) to be set to the one's complement of its original value; that is, all ones become zeros, and all zeros become ones.

e.g.	Before one's complement	-	01000 1100 1110 1111
	After one's complement	-	10111 0011 0001 0000

Clear Carry - CLC;

This instruction causes the Carry flag to be cleared.

Complement Carry - COMPC;

This instruction causes the state of the Carry flag to be complemented (i.e. reversed).

Unconditional Skip - SKIP;

This instruction causes the program to skip the next step, unconditionally.

Swap A or B - SWAPA; or SWAPB;

This instruction causes the contents of the top half (Bits 16 to 9) of the specified accumulator to be swapped with the contents of the bottom half (Bits 8 to 1) of the said accumulator. The sign bit is not affected.

Clear Sign of A or B - CLSA; or CLSB;

This instruction causes the sign bit (Bit 17) of the specified accumulator to be cleared (set to zero).

Complement Sign of A or B - COMPSA; or COMPSB;

This instruction causes the state of the sign bit (Bit 17) of the specified accumulator to be complemented (i.e. reversed).

Enter Switch Register into A or B - ESWRA; or ESWRB;

This instruction causes whatever is set on the Data Switches of the Control Panel to be loaded into the specified accumulator.

Mode 3 - contains 18 basic instructions which enable the programmer to perform tests on the Accumulator, Carry flag and Greater Than flag and to skip the next instruction depending on the results of the test. The group is sub-divided into two sections, the instructions of which cannot be mixed together, although six instructions (CLC, CLGT, CLA, CLB, COMPA and COMPB) appear in both sections. The Selection Tables for the two sub-sections now follow :-

SECTION 1

Mnemonics	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Octal Representation
	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
ANEG;	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	007600
BNEG;	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	0	0	006600
ANØ;	0	0	0	0	0	1	1	1	1	0	1	0	0	0	0	0	0	007500
BNØ;	0	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	0	006500
SKC;	0	0	0	0	0	1	1	0	1	0	0	1	0	0	0	0	0	006440
CLC;	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0	006020
SKGT;	0	0	0	0	0	1	1	0	1	0	0	0	0	1	0	0	0	006410
CLGT;	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	006004
CLA;	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1	0	007002
CLB;	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	006002
COMPA;	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	007001
COMPB;	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	006001

SECTION 2

	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
APOS;	0	0	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0	007200
BPOS;	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	006200
A Ø;	0	0	0	0	0	1	1	1	0	0	1	0	0	0	0	0	0	007100
B Ø;	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	006100
SKNC;	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	006040
CLC;	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0	006020
SKNGT;	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	006010
CLGT;	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	006004
CLA;	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1	0	007002
CLB;	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	006002
COMPA;	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	007001
COMPB;	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	006001

Fig. 3:21

The operation of each individual instruction specified by Bits 10 to 1 is described below, under the two Section headings :-

SECTION 1

Skip if A or B is negative - ANEG; or BNEG;

This instruction causes the next instruction to be skipped if the contents of the specified accumulator are negative (i.e. if the sign bit is set).

Skip if A or B is not zero - ANØ; or BNØ;

This instruction causes the next instruction to be skipped if the contents of the specified accumulator are not zero (i.e. if any of Bits 1 - 17 is set).

Skip if Carry - SKC;

This instruction causes the next instruction to be skipped if the Carry flag is set.

Clear Carry - CLC;

This instruction causes the Carry flag to be reset (cleared).

Skip if Greater Than - SKGT;

This instruction causes the next instruction to be skipped if the Greater Than flag is set.

Clear Greater Than - CLGT;

This instruction causes the Greater Than flag to be cleared.

Clear A or B - CLA; or CLB;

This instruction sets the specified accumulator (all 17 bits) to zeros.

Complement A or B - COMPA; or COMPB;

This instruction causes the specified accumulator (all 17 bits) to be set to the one's complement of its original value; that is, all ones become zeros, and all zeros become ones.

SECTION 2

Skip if A or B is positive - APOS; or BPOS;

This instruction causes the next instruction to be skipped if the contents of the specified accumulator are positive (i.e. if the sign bit is not set).

Skip if A or B is zero - AØ; or BØ;

This instruction causes the next instruction to be skipped if the contents of the specified accumulator (all 17 bits) are zero.

Skip if Not Carry - SKNC;

This instruction causes the next instruction to be skipped if the Carry flag is not set.

Clear Carry - CLC;

This instruction causes the Carry flag to be reset (cleared).

Skip if Not Greater Than - SKNGT;

The next instruction will be skipped if the Greater Than flag is not set.

Clear Greater Than - CLGT;

This instruction causes the Greater Than flag to be cleared.

Clear A or B - CLA; or CLB;

This instruction sets the specified accumulator (all 17 bits) to zeros.

Complement A or B - COMPA; or COMPB;

This instruction causes the specified accumulator (all 17 bits) to be set to the one's complement of its original value; that is, all ones become zeros, and all zeros become ones.

c) LITERAL (I/O, SHIFTS, etc.) INSTRUCTIONS

As all Input/Output instructions are deemed Privileged, this section describes first the Literal functions, which deal with multiple short shifts and rotates. These Literal functions have the following format in Machine Language :-

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Ø	Ø	Ø	Ø	Ø	Ø	Ø	A/B	1	L/R	S/R	Ø	A/L				

$\underbrace{17 \ldots 13}_{\text{OP. CODE (all zeros)}}$ $\underbrace{12\ 11}_{\text{MODE}}$ $\underbrace{4 \ldots 1}_{\text{No. of Shifts or Rotates (O to 15 Dec.)}}$

Fig. 3:22

The instruction word uses Bits 17-13 to specify the operation code (being all zeros for Literal functions), Bits 12 and 11 to specify the Mode (again both zeros), Bit 10 to specify the relevant accumulator, Bit 9 on to specify Special Shift, Bit 8 to specify Left or Right, Bit 7 to specify Shift or Rotate, Bit 6 being always zero, Bit 5 to specify Arithmetic or Logical and Bits 4 to 1 to indicate the number of shifts or rotates required (Ø to 15). These instructions are <u>not</u> micro-programmable.

Fig. 3:23 below shows the binary structure of each Literal instruction, but without showing the number of shifts or rotates required. (The number (Ø to 17 Octal) should be added in Octal.)

Mnemonics	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Octal Representation
	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	
SLAA△Ø;	0	0	0	0	0	0	0	1	1	1	1	0	1	NO.OF SHIFTS OR ROTATES				001720
SLAB△Ø;	0	0	0	0	0	0	0	0	1	1	1	0	1					000720
SRAA△Ø;	0	0	0	0	0	0	0	1	1	0	1	0	1					001520
SRAB△Ø;	0	0	0	0	0	0	0	0	1	0	1	0	1					000520
SLLA△Ø;	0	0	0	0	0	0	0	1	1	1	1	0	0					001700
SLLB△Ø;	0	0	0	0	0	0	0	0	1	1	1	0	0					000700
SRLA△Ø;	0	0	0	0	0	0	0	1	1	0	1	0	0					001500
SRLB△Ø;	0	0	0	0	0	0	0	0	1	0	1	0	0					000500
RLMA△Ø;	0	0	0	0	0	0	0	1	1	1	0	0	0					001600
RLMB△Ø;	0	0	0	0	0	0	0	0	1	1	0	0	0					000600
RRMA△Ø;	0	0	0	0	0	0	0	1	1	0	0	0	0					001400
RRMB△Ø;	0	0	0	0	0	0	0	0	1	0	0	0	0					000400

Fig. 3:23

The operation of each individual instruction is briefly described below :-

Shift Left Arithmetic A or B - SLAA or SLAB

Bits 16 to 1 of the specified accumulator will be shifted left the specified number of places. The sign bit (Bit 17) remains unchanged.

Any bit shifted off the top of the register sets the Carry flag but is otherwise lost. A zero will be inserted in Bit 1 on each left shift.

```
e.g.    SLAAΔ4; of  0  0110  1100  0110  0101
             gives  0  1100  0110  0101  0000  (with Carry Flag set)
```

Shift Right Arithmetic A or B - SRAA or SRAB

Bits 16 to 1 of the specified accumulator will be shifted right the specified number of places. The sign bit (Bit 17) remains unchanged, but is propagated into the Bit 16 position on each right shift.

Any bit shifted off the bottom of the register sets the Carry flag but is otherwise lost.

```
e.g.   SRAAΔ4;  of  1  0110  1100  0110  0101
             gives  1  1111  0110  1100  0110   (With Carry flag set)
```

Shift Left Logical A or B - SLLA or SLLB

Bits 16 to 1 of the specified accumulator will be shifted left the specified number of places. The sign bit (Bit 17) remains unchanged.

Any bit shifted off the top of the register sets the Carry flag but is otherwise lost. A zero will be inserted in Bit 1 on each left shift.

```
e.g.   SLLBΔ2;  of  0  0110  1100  0110  0101
             gives  0  1011  0001  1001  0100   (With Carry flag set)
```

Shift Right Logical A or B - SRLA or SRLB

Bits 16 to 1 of the specified accumulator will be shifted right the specified number of places. The sign bit (Bit 17) remains unchanged.

Any bit shifted off the bottom of the register sets the Carry flag but is otherwise lost. A zero will be inserted in Bit 16 on each right shift.

```
e.g.   SRLAΔ3;  of  0  0110  1100  0110  0101
             gives  0  0000  1101  1000  1100    (With Carry flag set)
```

Rotate Left Multiple A or B - RLMA or RLMB

Bits 16 to 1 of the specified accumulator will be rotated left the specified number of places, with the sign bit (Bit 17) remaining unchanged and the Carry flag being unaffected. Bits 15 to 1 will be left shifted once on each rotate, with the figure which was in Bit 16 re-entering the Accumulator at Bit 1. In other words, Bits 1 to 16 of the specified accumulator are treated as a closed loop, and what is commonly called a circular shift is performed for each rotate, in that any bit rotated off the left end will re-appear at the right end.

```
e.g.   RLMAΔ2; of    0  0110  1100  0110  0101
            gives    0  1011  0001  1001  0101
```

Rotate Right Multiple A or B - RRMA or RRMB

Bits 16 to 1 of the specified accumulator will be rotated right the specified number of places, with the sign bit (Bit 17) remaining unchanged and the Carry flag being unaffected. Bits 16 to 2 will be right shifted once on each rotate, with the figure which was in Bit 1 re-entering the Accumulator at Bit 16. In other words, Bits 1 to 16 of the specified accumulator are treated as a closed loop, and what is commonly called a circular shift is performed for each rotate, in that any bit rotated off the right end will re-appear at the left end.

```
e.g.   RRMBΔ3; of    0  0110  1100 0110  0101
            gives    0  1010  1101 1000  1100
```

INPUT/OUTPUT INSTRUCTIONS

These are a set of program instructions which, like the majority of the Mode 0 Register Instructions, are 'Privileged', that is not used in general programming but only by the system software. There now follows a description of the Input/Output instructions, which control all transfers of data to and from the peripherals and also perform various operations within the processor. They provide the following general capabilities :-

a) Fix the state of the Busy and Done flags
b) Test the state of the Busy and Done flags
c) Enter data from a specified device into the A or B Registers
d) Output data to a specified device from the A or B Registers.

I/O instructions are recognised by the computer when the four most significant bits of the instruction word are 0000 and Bit 13 is a 1 (Octal Code 01). Bits 6 to 1 select the device that is to respond to the instruction; the format thus allows for 64 codes. In all I/O instructions Bits 11 to 7 specify the complete function to be performed, bits 10 and 11 either controlling or sensing Busy and Done, as shown under 'Funct.' in the Selection Chart (Fig.3:24) on the next page. If Bits 7 to 9 are all set (Mode 111) there is no transfer, and Bits 11 and 10 then specify a skip condition. Bit 12, where relevant, specifies the A or B register (A=1, B=Ø).

	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1		
	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
Mnemonics	OP. CODE					A/B	FUNCT.		MODE			DEVICE CODE							
STT	0	0	0	0	1		0	1										Set Busy. Clear Done	
STOP	0	0	0	0	1		1	0										Clear Busy. Clear Done	
IOP	0	0	0	0	1		1	1										Input/Output Pulse	
	0	0	0	0	1				0	0	0							Allows Functions above	
DIP1A	0	0	0	0	1	1			0	0	1							DATI 1A	
DIP1B	0	0	0	0	1				0	0	1							DATI 1B	
DIP2A	0	0	0	0	1	1			0	1	0							DATI 2A	
DIP2B	0	0	0	0	1				0	1	0							DATI 2B	
DIP3A	0	0	0	0	1	1			0	1	1							DATI 3A	
DIP3B	0	0	0	0	1				0	1	1							DATI 3B	Can be used with any of above functions
DOP1A	0	0	0	0	1	1			1	0	0							DATO 1A	
DOP1B	0	0	0	0	1				1	0	0							DATO 1B	
DOP2A	0	0	0	0	1	1			1	0	1							DATO 2A	
DOP2B	0	0	0	0	1				1	0	1							DATO 2B	
DOP3A	0	0	0	0	1	1			1	1	0							DATO 3A	
DOP3B	0	0	0	0	1				1	1	0							DATO 3B	
SKB	0	0	0	0	1		0	0	1	1	1							Skip if Busy	
SKNB	0	0	0	0	1		0	1	1	1	1							Skip if not Busy	
SKD	0	0	0	0	1		1	0	1	1	1							Skip if Done	
SKND	0	0	0	0	1		1	1	1	1	1							Skip if not Done	

Fig. 3:24

Operation Code

Input/Output instructions are recognised by the computer when Bits 17 to 13 are set to the Octal Code 01.

Accumulator Indicator - Bit 12

Denotes to which accumulator the instruction refers

Accumulator A	=	1
Accumulator B	=	Ø

Function - Bits 11 and 10

When any Mode except 111 is set, the functions are as follows :-

00	=	No operation	
01	=	Set Busy, Clear Done	(to start the device)
10	=	Clear Busy, Clear Done	(to idle the device)
11	=	Input/Output Pulse	(the effect, if any, depends on the device).

The functions as interpreted when Mode 111 (Skip Mode) is set, are shown with this Mode.

MODES - Bits 9-7

Mode 000

This has no input/output transfer, but just enables any of the functions shown previously.

Mode 001 - DIP1A or DIP1B

Enables input from Register 1 of the specified device (each peripheral can have up to three registers or buffers) into Accumulator A or B, as specified, and can also enable any Function shown previously.

Mode 010 - DIP2A or DIP2B

Enables input from Register 2 of the specified device; used to inspect Status codes, and can also enable any Function shown previously.

Mode 011 - DIP3A or DIP3B

Enables input from Register 3 of the specified device into Accumulator A or B, as specified, and can also enable any Function shown previously.

Mode 100 - DOP1A or DOP1B

Enables output to Register 1 of the specified device from Accumulator A or B, as specified, and can also enable any Function shown previously.

Mode 101 - DOP2A or DOP2B

Enables output to Register 2 of the specified device from Accumulator A or B, as specified, and can also enable any Function shown previously.

Mode 110 - DOP3A or DOP3B

Enables output to Register 3 of the specified device from Accumulator A or B, as specified, and can also enable any Function shown previously.

Mode 111 - Skip Mode

If this Mode is set there is no input/output transfer and the Function bits (11 and 10) then specify a skip condition.

e.g.	Function 00	=	Skip if Busy
	Function 01	=	Skip if Not Busy
	Function 10	=	Skip if Done
	Function 11	=	Skip if Not Done

Device Code

There can be 64 different device codes (00-77 in Octal) of which 63 are used to address devices, the code 00 not being used. A table in Figure 3:25 on the next page lists all devices for which codes have been assigned. The addresses have been grouped into fixed blocks to cover all machine variations, to prevent inadvertent misdirection of data between devices in test programs or board changing. Devices must all carry serial numbers on program descriptions and listing, and all programmers must notify the Factory in writing as soon as possible which devices have which codes, and also the order of priorities as required.

Alpha/Numeric Displays have their own addresses, ranging normally from 60 to 67. When required, they are paired with Keyboards as follows :-

Keyboard 20 with A/N Display 60
Keyboard 21 with A/N Display 61

etc.

I/O DEVICE ADDRESSES

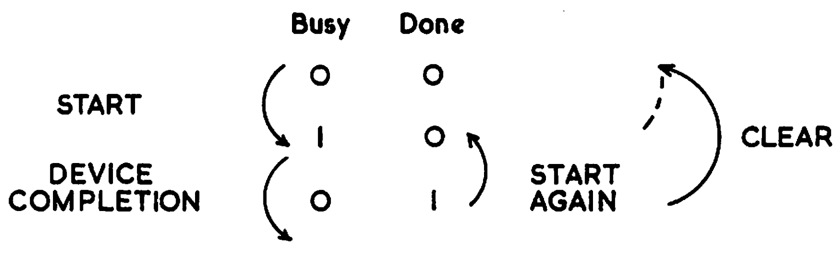
00	Not Used			
01	Mag. Stripe Reader 1	Front Feed		
02		Rear Feed		
03	Mag. Stripe Reader 2	Front Feed		
04		Rear Feed		
05	Mag. Stripe Reader 3	Front Feed		
06		Rear Feed		
07	Spare			
10	PE Reader 2	or	Mark Senser 1	
11	PE Reader 1	or	Mark Senser 2	
12			Mark Senser 3	
13			Mark Senser 4	
14	Single Shot/Hopper Reader(80 col)		or	" 5
15	" " " " e.p.c.		or	" 6
16	" " " " "		or	" 7
17	" " " " "		or	" 8
20	Alpha-Numeric K/B	1	10 Key K/B	1
21	" " "	2	"	2
22	" " "	3	"	3
23	" " "	4	"	4
24	" " "	5	"	5
25	" " "	6	"	6
26	" " "	7	"	7
27	" " "	8	"	8
30	Line Printer	1		
31	Line Printer	2		
32	Tape Punch	2		
33	Tape Punch	1		
34	High Speed Serial Printer (Dot) 1			
35	High Speed Serial Printer (Dot) 2			
36	----------------Spare----------------			
37	----------------Spare----------------			

40	Serialiser (Modem)TX Cathode Ray Display	1	
41	" " "	2	
42	" " "	3	
43	" " "	4	
44	" " "	5	
45	" " "	6	
46	" " "	7	
47	" " "	8	
50	Serialiser (Modem)RX Cathode Ray Keyboard	1	
51	" " "	2	
52	" " "	3	
53	" " "	4	
54	" " "	5	
55	" " "	6	
56	" " "	7	
57	" " "	8	
60	I/O Writer	4 I/P	A/N Display or Tally Roll Printers
61	"	4 O/P	
62	"	3 I/P	"
63	"	3 O/P	"
64	"	2 I/P	"
65	"	2 O/P	"
66	"	1 I/P	"
67	"	1 O/P	"
70	Disc	1	
71	Disc	2	
72	----------Spare------------		
73	----------Spare------------		
74	Mag Tape Handler 1		
75	Mag Tape Handler 2		
76	Drum/Disc (Fast Access) 1		
77	Drum/Disc (Fast Access) 2		

Fig. 3:25

Input/Output (continued)

Every peripheral device has up to three buffer registers, an Interrupt Disable Flag, Busy and Done Flags and a 6-bit device selection network. This selection network decodes Bits 1-6 of the Input/Output Instruction (which contain the device address), thus ensuring that only the correct device responds to signals sent by the processor over the in-out bus. The Busy and Done Flags together denote the basic state of the device. When both are clear, the device is idle. The overall sequence of Busy and Done states is determined by both the program and the internal operation of the device :-



The Data-in or data-out instruction that the program gives in response to the setting of Done can also restart the device. When all data has been transferred, the program generally clears Done so that the device neither requests further interrupts nor appears to be in use. (Busy and Done both set is a meaningless situation).

Any device whose Interrupt Disable flag is set, cannot cause an interrupt to start and is therefore regarded by the program as being of low priority. The Interrupt Disable flags are used in setting up a priority structure which enables higher priority devices to interrupt an interrupt already in progress. This priority is determined by the use of a mask which controls the states of the Interrupt Disable flags in the different devices (as previously described under Mask Out, see Page 3:19).

Each peripheral device can have up to three buffer registers, as explained on previous page. DATI 2 Input (DIP 2A or DIP 2B) inspects the Status Codes of all devices except disc. Status codes are codes allocated to indicate conditions of the device being addressed, and appear in Bits 8 - 1 as shown below, reading from left to right in descending order. Parity false condition is reset by the Input/Output Pulse in all cases except the Trend Reader.

STATUS CODES

Device	Status	Code
Card Reader	No Card	00000001
Card Reader	Parity Odd	00001000
Displays	Power Monitor	00000001
Fast Serial Printer	Power Monitor	00000001
IBM (see note below)	Parity Fail	10000000
Keyboards	Parity Error	00000010
Keyboards	Finger Trouble	00000100
Line Printer	Off Line	00000001
Line Printer	Buffer not Ready	00000010
Line Printer	Print cycle in progress	00000100
Line Printer	Paper feeding	00001000
Line Printer	Data Channel Mode	00010000
Mag. Stripe Card Handler	Last Line	00000001
Mag. Stripe Card Handler	Card jammed	00000010
Mag. Stripe Card Handler	Parity fail (Read only)	00000100
Mag. Stripe Card Handler	Data late (read or write)	00001000
Mag. Stripe Card Handler	Card not in Hopper	00010000
Modems	Do not transmit	00000001
Modems	Do not transmit or receive	00000010
Modems	Data link not connected	00000100
Modems	Parity Odd	00001000
Modems	Receiver Idling	00010000
Paper Tape Reader	Latch Open	00000001
Paper Tape Reader	No Tape	00000010
Paper Tape Reader	Parity Odd	00001000
Punch	Card position	00000001
Punch	Out of Tape	00000100
Punch	Check-back-failure	00000010
Single Shot	No Card	00000001
Single Shot	Parity Odd	00001000
Tally Roll Printer	Power Monitor	00000001
VDU	Parity Error	00001000

Note :- Any parity failure on Input or Output will be detected and shown as a status failure in the Status Register of the IBM Input only.

The Status Codes referring to 'No Tape' in the Trend Reader and 'Out of Tape' in the Punch will be available only when decided by the Technical design/development/production departments. Please refer to Pre-production Engineers for news of latest positions on these Status Codes.

For Disc Status Codes, please see Page 4:37.