4. INSTRUCTIONS

Instructions direct the input, processing and output of information. They specify the actions to be carried out by the computer, and direct the sequence of events.

The general form of CREDIT instructions is:-

```
    [STATEMENT-IDENTIFIER]    INSTRUCTION-MNEMONIC    OPERAND-1[,OPERAND-2...]
```

STATEMENT IDENTIFIER - this identifies a point within the program and is used in branch and entry instructions; a statement identifier need not be on the same line as the statement, e.g.

```
    FRED1      ADD       A,B        this statement is identified as FRED1
```

could also be written on two lines, e.g.

```
    FRED1
               ADD       A,B
```

INSTRUCTION MNEMONIC - this specifies the basic operation to be performed by the instruction. There are nine groups of instructions, and the instruction mnemonic must be derived from one of these groups; see appendix A for a list of instructions and categories. In the above example the INSTRUCTION MNEMONIC was ADD, one of the arithmetic group of instructions.

OPERANDS - these contain the operational part of the instructions, their significance being different for each instruction. Each instruction described here and in M04 refers to operands from left to right as OPERAND-1, OPERAND-2 etc.

The groups of instructions available in CREDIT are as follows:

Arithmetic
Branch
Input/Output
Logical
Scheduling
Storage Control
String
Subroutine control
Format I/O control

GROUPS OF INSTRUCTIONS

- ARITHMETIC INSTRUCTIONS
- LOGICAL INSTRUCTIONS
- STRING INSTRUCTIONS
- BRANCH INSTRUCTIONS
- SUBROUTINE INSTRUCTIONS
- INPUT/OUTPUT INSTRUCTIONS
- SCHEDULING INSTRUCTIONS
- STORAGE CONTROL INSTRUCTION

4.1 Arithmetic instructions

These have two operands, and consist of the following instruction mnemonics:-

ADD	Add
CMP	Compare
DIV	Divide
DVR	Divide rounded
MOVE	Move (conversions)
MUL	Multiply
SUB	Subtract

The arithmetic instructions ADD, SUB, MUL, DIV and DVR operate on either BCD or BIN data items or arrays. Both operands must be of the same type, e.g. both BIN or both BCD.

The CMP instruction operates on BCD, BIN or STRG data items or arrays. Both operands must be of the same type.

The MOVE instruction can be used with BIN, BCD or STRG data items or arrays, and the operands need not be of the same type.

After the execution of most CREDIT instructions the status will be held in a special register by the Interpreter, called the Condition Register (CR). This can be used to determine the path the program takes on different conditions.

Arithmetic instructions will cause the Condition Register to be set as shown in the table below.

Value	Meaning
0	Zero result
1	Positive result
2	Negative result
3	Arithmetic overflow occurred

ADD INSTRUCTION

```
                             IDENT    MAIN
                             DDIV
                             TERM     T0
                             TWB      TB1
                             START    TAGO

                  TB1        BLK
10                CATOT      BOOL     FALSE
11                EOTAPE     BOOL     FALSE
10                INDEX      BIN      'O'
11                INLEN      BIN      'O'
12                LAMP1      BIN      X'0200'
13                LAMP9      BIN      X'0100'
14                SUBACC     BCD      8
15                TWORK1     BCD      8
16                ARRAY      BCD1     (5),10D'O'
19                DMSTR      STRG     1C'  '

                             PDIV
                             ENTRY    TAGO

                  TAGO
02 14 15                     ADD      SUBACC, TWORK1

02 16 10 15                  ADD      ARRAY (INDEX), TWORK1
```

4.1.1 The Add instruction (ADD)

The ADD instruction adds operand-2 to operand-1 and stores the result in operand-1. Execution of this instruction will affect the Condition Register, as shown in the table in section 4.1.

Examples of the ADD instruction

```
        ADD       ACC,DEP
```

This increases the contents of the data item ACC by the contents of the data item DEP.

```
        ADD       LOOP,=W'1'
```

This increases the contents of the binary data item LOOP by one

```
        ADD       CASH(TID,USER),DEP
```

This increases the element in array CASH referenced by the subscripts TID and USER by the amount held in the data item DEP.

4.1.2 The Subtract instruction (SUB)

The SUB instruction subtracts the contents of operand-2 from the contents of operand-1; execution of this instruction will alter the Condition Register as shown on page 4.1.1.

Examples of the SUB instruction

```
        SUB       LOOP,=W'1'
```

If LOOP had an initial value of 10 then after execution of this statement it would contain 9.

```
        SUB       TEST,=W'-1'
```

If test held an initial value of 5 then after this statement had been executed it would hold the value 6.

INSTRUCTION	PAGE IN M04
ADD	1.4.24
SUB	1.4.149

4.1.3 The Divide instruction (DIV)

The DIV instruction divides the contents of operand-1 by the contents of operand-2 and stores the result in operand-1, any remainder being ignored. Division by zero results in overflow, thus giving a value of 3 in the CR. The Condition Register contents are described on page 4.1.1.

Example of the DIV instruction

```
        DIV       COM,=W'100'
```

This divides the data item COM by 100, if COM had an initial value of 378, then after this instruction COM would have the value 3.

```
         COM/100          = 3.78
         Rounded down     = 3

         Result in COM    = 3
```

4.1.4 The Divide Rounded instruction (DVR)

The DVR instruction divides the contents of operand-1 by the contents of operand-2; 0.5 is then added to the result and the rounded down result stored in operand-1. The Condition Register is set as described on page 4.1.1

Example of the DVR instruction

```
        DVR       COM,=W'100'
```

If the data item COM had the initial value 378 then after the division the result would be 4, as shown below.

```
        COM/100        = 3.78
        + 0.5          = 4.28
        Rounded down   = 4

        Result in COM = 4
```

4.1.5 The Multiply instruction (MUL)

The MUL instruction multiplies the contents of operand-1 by the contents of operand-2 and stores the result in operand-1. The CR is set as described on page 4.1.1.

Example of the MUL instruction

```
        MUL       AMM,XRATE
```

The data item AMM is multiplied by the data item XRATE. If the initial value of AMM was 1700 and XRATE 450 then after this statement had been executed AMM would contain the value 765000. If the receiving data item is not large enough to hold the result, its contents will be undefined and the CR will be set to to overflow (value 3).

INSTRUCTION	PAGE IN M04
DIV	1.4.68
DVR	1.4.91
MUL	1.4.129

4.1.6 The Compare instruction (CMP)

The CMP instruction compares two data items of the same type for similarity. It sets the condition register to one of the values shown below, according to the relationship between the two data items.

When the two data items are of different lengths, the comparison will be executed as follows:

- For string data items the shortest data item will be extended (by the Interpreter) with blank characters (X'20') from the right.
- For decimal data items the shortest data item will be extended (by the Interpreter) with zero digits (X'0').

The values held in the Condition Register after execution of this instruction are:-

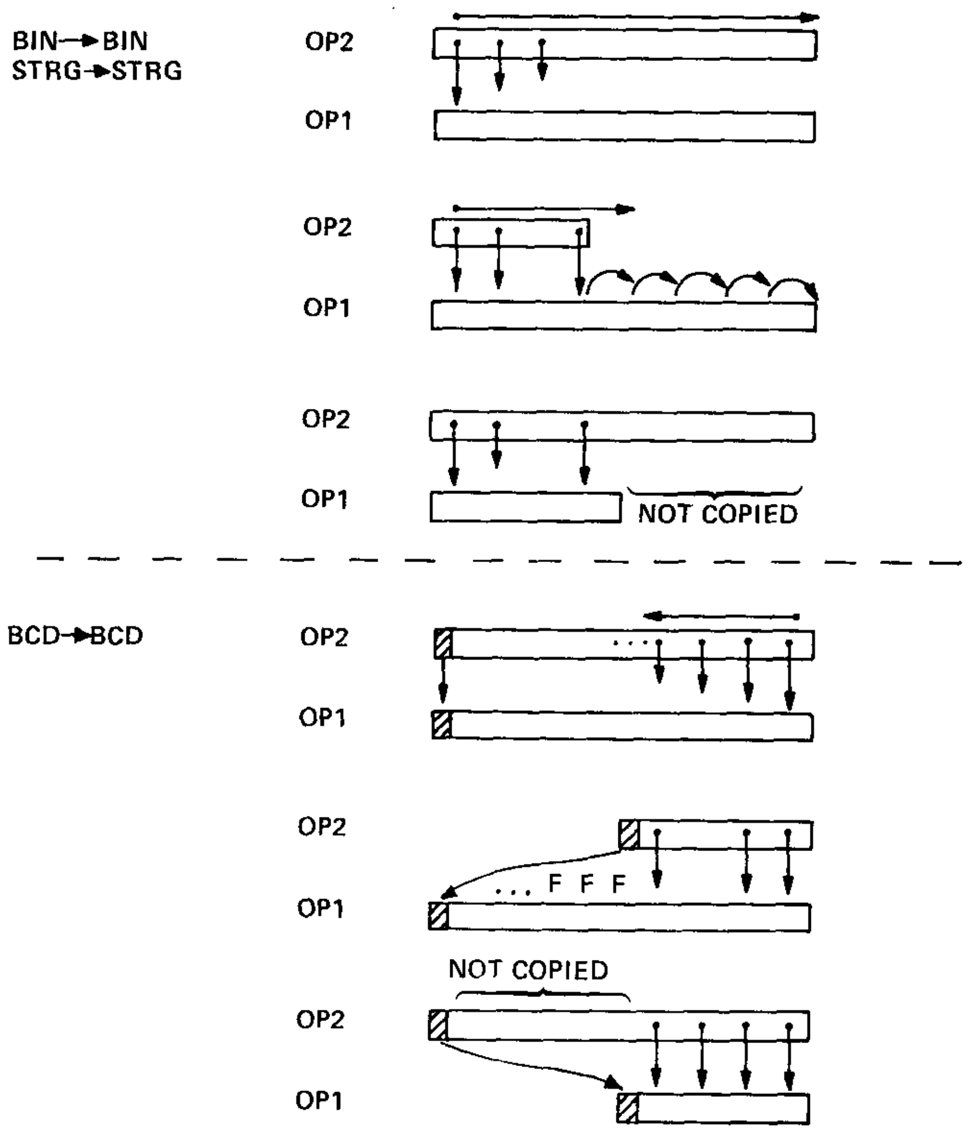
VALUE	MEANING
0	Operand-1 = Operand-2
1	Operand-1 > Operand-2
2	Operand-1 < Operand-2

Examples of the CMP instruction

```
         CMP      SPBINW1,SPBINW2          Two items of like type

         CMP      SPBINW3,=W'97'           Comparison with a constant
         CMP      SPINPUT,=C'YES'

CON1     EQU      W'97'
         CMP      SPBINW3,CON1             Use of an EQU constant
```

INSTRUCTION	PAGE IN M04
CMP	1.4.62

MOVE OP1, OP2


BIN→BIN
STRG→STRG
OP2
OP1
OP2
OP1
OP2
OP1
NOT COPIED
BCD→BCD
OP2
OP1
OP2
. . . F F F
OP1
NOT COPIED
OP2
OP1

4.1.7 The Move instruction (MOVE)

The MOVE instruction moves the contents of operand-2 to operand-1. The operands can be of type BIN, BCD or STRG, though transfer from BIN to STRG or STRG to BIN is not permitted.

The MOVE can be used for transferring numeric information only between the data types linked by arrows in the diagram below.

```
BIN <---> BIN <---> BCD <---> BCD <---> STRG <---> STRG
```

The rules for moving of data items is given on pages 1.4.127-128 of M04. In summary, moving to a shorter data item causes truncation of information, and moving to a longer data item results in padding.

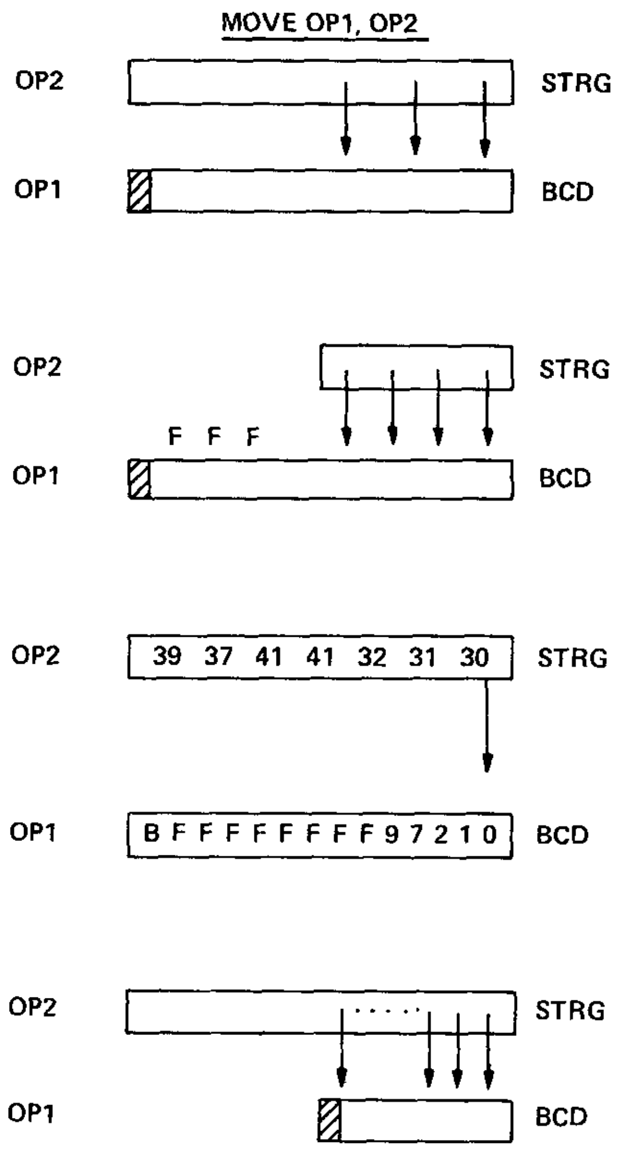
Examples of the MOVE instruction

```
         MOVE      OUT,=C'PLEASE ENTER USER CODE'
```

If OUT has been defined as STRG with a length of twenty-two characters then the character string would be transferred to OUT. If OUT were longer than twenty two characters then the last character in the string would be repeated. For example, if OUT had been defined as length twenty-five, then after this instruction had been executed it would contain "PLEASE ENTER USER CODEEEE". If OUT had been defined as length ten, then it would contain "PLEASE ENT", only the left hand ten characters being transferred.

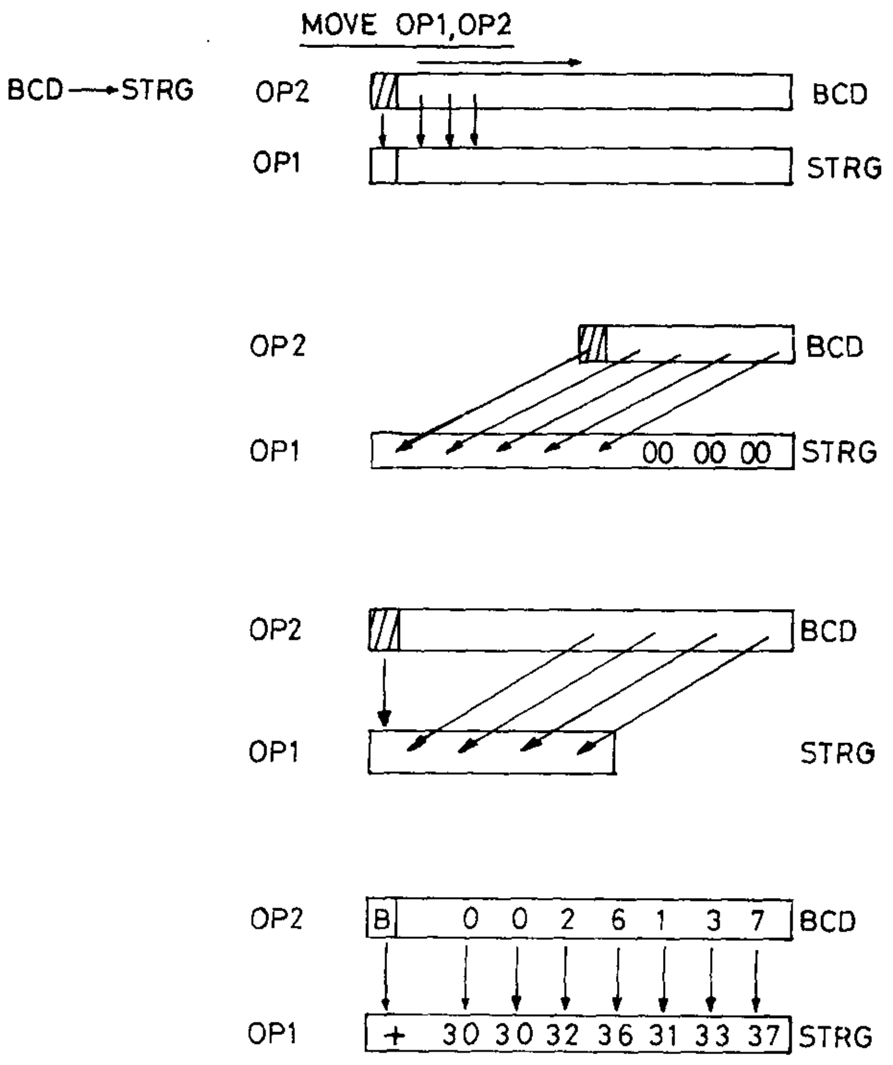
INSTRUCTION	PAGE IN M04
MOVE	1.4.127

STRG→BCD


MOVE OP1, OP2
OP2
STRG
OP1
BCD
OP2
STRG
F F F
OP1
BCD
OP2
39 37 41 41 32 31 30
STRG
OP1
B F F F F F F F F 9 7 2 1 0
BCD
OP2
STRG
OP1
BCD

```
         MOVE      OUTX,=C'PLEASE ENTER YEAR E.G. 1979 : '
```

If, in the above example, OUTX had been defined as BCD then it would contain '1979' as non-numeric characters are not transferred from STRG to BCD. Note that after execution the contents of OUTX will have been right justified, have the sign digit set and all unused digits will be set to X'F' (X'F' is the null digit for BCD items). If a transfer of a number to either a BCD or BIN data item is requested, and the number is too large to be held by that data item then its contents will be uncertain and the Condition Register will be set to 3 (Overflow).


MOVE OP1,OP2
BCD → STRG
OP2
BCD
OP1
STRG
OP2
BCD
OP1
00 00 00
STRG
OP2
BCD
OP1
STRG
OP2
B 0 0 2 6 1 3 7
BCD
OP1
+ 30 30 32 36 31 33 37
STRG

```
MOVE        FLDA,NUMB
```

If FLDA has been declared as a STRG data item, four bytes long, and NUMB is a BCD data item containing the value +123456, then the result in FLDA will be

```
+456
```

since moving from BCD to STRG always results in the sign being moved first. The remaining digits are moved, and converted, from left to right, but in this case the receiving field is too short, hence only the rightmost digits are transferred.

4.2 Logical instructions

These are single operand instructions and allow logical operations on boolean data items. Boolean data items are used for holding such things as status flags. Note that the Condition Register will be set to the previous contents of the data item after execution of a logical instruction.

The available instructions are

CLEAR	Clear a boolean data item (result binary zero)
INV	Invert a data item (reverse its state)
SET	Set a data item (result binary one)
TEST	Compare with zero (false) and set condition register

Examples of logical instructions

```
CLEAR     FLAG
```

This sets the boolean data item FLAG to FALSE (zero)

```
INV       FLAG
```

The state of FLAG is reversed - if it was FALSE it will now be TRUE

INSTRUCTION	PAGE IN M04
CLEAR	1.4.61
INV	1.4.117
SET	1.4.146
TEST	1.4.155

4.3 String instructions

These instructions are for the manipulation of character strings.

The available instructions are

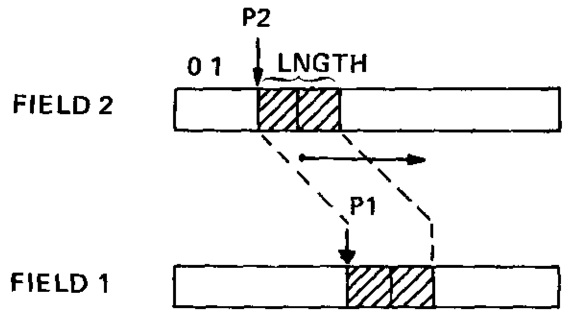
COPY
MATCH
INSERT
DELETE
XCOPY

These instructions operate on string data items, with two exceptions
. The XCOPY command can be used with both STRG and BCD data items.
. The COPY command can be used with BIN, BCD and STRG data items.

The string handling commands have two different types of operand.
. Character strings (or BIN and BCD data items in the case of the two exceptions listed above).
. Pointers and lengths, held in binary data items.

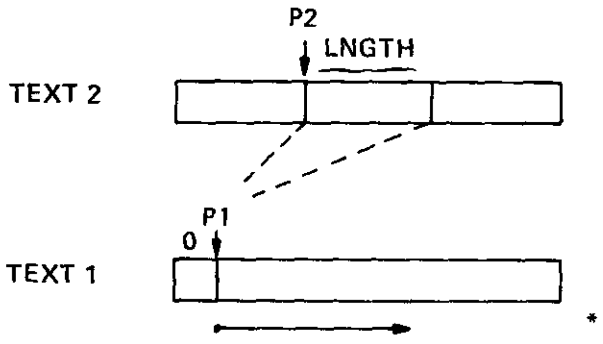
COPY INSTRUCTION

COPY FIELD1,PL,LNGTH,FIELD2,P2



INSERT INSTRUCTION

INSRT TEXT1, P1, LNGTH, TEXT2, P2



ORIGINAL CONTENTS SHIFTED TO THE RIGHT

* COND.REG.=3 IF NON-SPACE
OR NON-ZERO
CHARACTER SHIFTED OUT

4.3.1 The Copy instruction (COPY)

The COPY instruction is used to move a number of decimal digits or bytes from one data item to another <u>of the same type</u>. Both data items must be STRG or both must be BCD or both must be BIN.

The instruction format is:-

```
        COPY      Dest, Start, No., Source, Start-2
```

Dest	- is the data item which is to have information copied into it.
Start	- is a binary data item containing a pointer to the position in Dest where the copied information is to begin.
No.	- is a binary data item containing the number of characters (bytes) or decimal digits (half bytes) to be copied from Source to Dest.
Source	- is the data item, part or all of which will be copied into Dest.
Start-2	- is a binary data item containing a pointer to the start of the information in Source that is to be copied into Dest.

The pointers (Start and Start-2) assume that the first byte location is zero, so to access the second byte or digit the pointer must have a value of one.

Example of the COPY instruction

```
        MOVE      S1,=W'0'
        MOVE      S2,=W'4'
        MOVE      S3,=W'1'
        COPY      DEST,S1,S2,SRC,S3
```

If SRC had been defined as a STRG data item ten bytes long, containing the string "XCURRENCY", and DEST as a STRG data item four bytes long, then after execution DEST would contain "CURR".

If SRC had been defined as a BCD data item ten digits long, containing the decimal number "523012350", and DEST as a BCD data item four digits long, then after execution DEST would contain "2301". Note that the sign position in a BCD data item can be changed by the program, by use of this instruction. The Condition Register is not affected by the execution of this instruction.

INSTRUCTION	PAGE IN M04
COPY	1.4.63

4.3.2 The Extended Copy instruction (XCOPY)

The XCOPY instruction moves bytes between any non-boolean data items. It always copies at byte level, regardless of data types.

The instruction format is:-

```
    XCOPY     Dest, Start, No., Source, Start-2
```

Dest	- is the data item which is to have information copied into it.
Start	- is a binary data item containing a pointer to the position in Dest where the copied information is to begin.
No.	- is a binary data item containing the number of characters to be copied from Source to Dest.
Source	- is the data item, part or all of which will be copied into Dest.
Start-2	- is a binary data item containing a pointer to the start of the information in source that is to be copied into dest.

The pointers (Start and Start-2) assume that the first byte location is zero, so to access the second byte or digit the pointer must have a value of one.

Example of the XCOPY instruction

```
         MOVE      S1,=W'0'
         MOVE      S2,=W'4'
         MOVE      S3,=W'1'
         XCOPY     DEST,S1,S2,SRC,S3
```

SRC has been defined as a STRG data item ten bytes long and contains the string 'ABCDEFGHI', and DEST as a BCD data item eight digits long. Then after execution of this statement DEST will contain '42434445', the hexadecimal equivalent of the character string 'BCDE'.

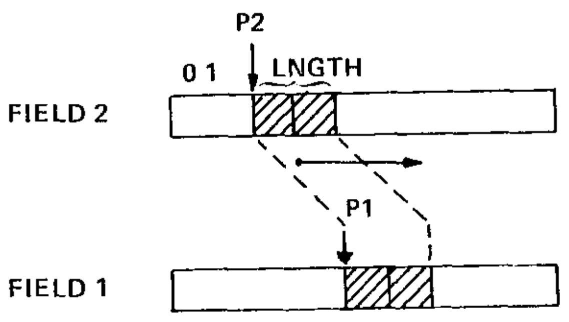
If SRC had been defined as a BCD data item fourteen digits long and contained the hexadecimal characters '24435552525553', and DEST as a STRG data item four characters long, then after execution of this instruction DEST would contain 'CURR'.

The Condition Register is not affected by the execution of this instruction.

INSTRUCTION	PAGE IN MQ4
XCOPY	1.4.173

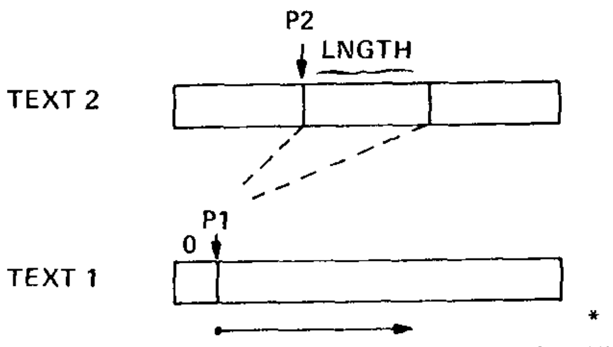
COPY INSTRUCTION

COPY FIELD1,PL,LNGTH,FIELD2,P2



INSERT INSTRUCTION

INSRT TEXT1, P1, LNGTH, TEXT2, P2



ORIGINAL CONTENTS SHIFTED TO THE RIGHT

* COND.REG.=3 IF NON-SPACE
OR NON-ZERO
CHARACTER SHIFTED OUT

4.3.3 The Insert instruction (INSRT)

The INSRT instruction is used to insert a character string into an existing character string, the existing contents being shifted to the right to produce the required space.

The instruction format is:-

```
          INSRT      Dest, Start, No., Source, Start-2
```

Dest	- is the data item which is to have information inserted into it.
Start	- is a binary data item containing a pointer to the position in Dest where the inserted information is to commence.
No.	- is a binary data item containing the number of characters from Source to be inserted into Dest.
Source	- is the data item, part or all of which will be inserted into Dest
Start-2	- is a binary data item containing a pointer to the start of the information in Source that is to be inserted into Dest.

The pointers (Start and Start-2) assume that the first byte location is zero, so to access the second byte or digit the pointer must have a value of one.

If a non-space or non-zero character is shifted out of Dest, the Condition Register will be set to 3 (overflow). Each character shifted out of the dataitem is lost.

Example of the INSRT instruction

```
          MOVE       S1,=W'5'
          MOVE       S2,=W'4'
          MOVE       S3,=W'4'
          INSRT      DEST,S1,S2,SRC,S3
```

If the initial contents of the string data item DEST was 'CODE:=N/A ' and the contents of SRC was '23456789' then after this operation DEST will contain 'CODE:=6789'. Note that the previous contents have been shifted to the right and as the field length was only ten characters the last four are lost. One data item could be saved by writing the instruction in the form shown below:-

```
          INSRT      DEST,S1,S2,SRC,S2
```

If the initial contents of DEST had been 'ABCDEFGHIJKL' and SRC contained '23456789' then after execution of this statement DEST would contain 'ABCDE6789FGHI'.

The details of this operation are shown below:

INSTRUCTION	PAGE IN M04
INSRT	1.4.116

DEST

ABCDEFGHIJKLM

SRC

23456789

Move four spaces into DEST

ABCDE   FGHI

take four characters from SRC starting at at character position four (the first character position is zero)

6789

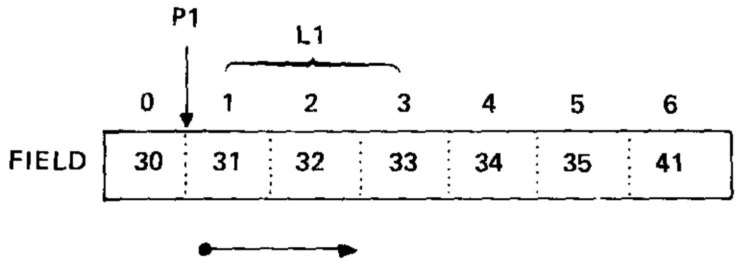
Now put the two parts together

ABCDE6789FGHI

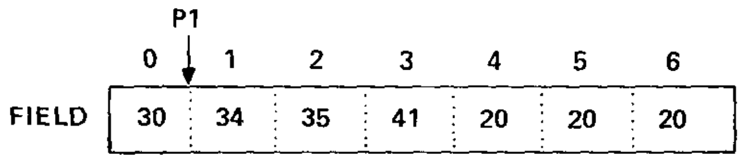
If a non-blank or non-zero character is lost off the end of the receiving field then the Condition Register will be set to 3 (overflow).

DELETE INSTRUCTION

DLETE FIELD, P1, L1



RESULT AFTER EXECUTION



4.3.4 The Delete instruction (DLETE)

The DLETE instruction is used to remove characters from a STRG data item, the characters remaining to the right of the deletion are then shifted to the left to fill the gap caused by the deleted characters, and spaces are used to fill from the right.

The Condition Register is not affected by the execution of this instruction.

The instruction format is:-

```
      DLETE     String, Start, No.
```

String	- is the character string which contains the item to be deleted
Start	- this is a binary data item giving the character position for the deletion to begin
No.	- this is a binary data item and contains the number of characters to be deleted

The pointer (Start) assumes that the first byte location is zero, so to access the second byte the pointer must have a value of one.

Example of the DLETE instruction

```
      MOVE      S1,=W'6'
      MOVE      S2,=W'4'
      DLETE     DEST,S1,S2
```

If the initial contents of DEST was 'SMITH MRS PAT', after the above section of program had been executed DEST would contain 'SMITH PAT   '.

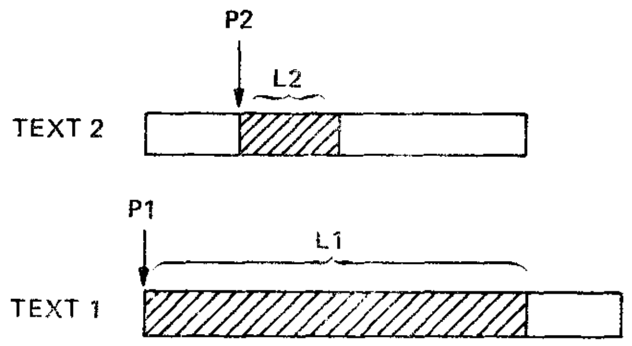
The details of this operation are shown below

```
      Delete characters           'SMITH ....PAT'      (. is deleted char)

      Move remaining characters   'SMITH PAT....'

      Fill from right with spaces 'SMITH PAT    '
```

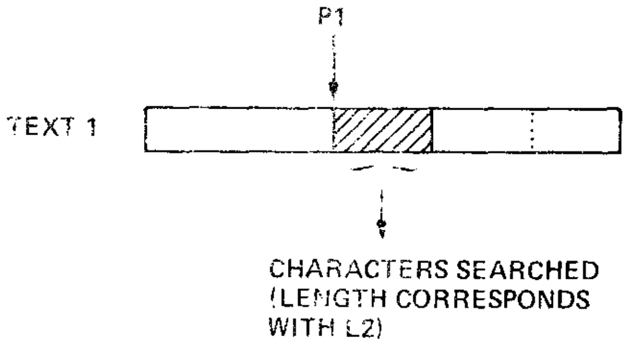
INSTRUCTION	PAGE IN M04
DLETE	1.4.69

MATCH INSTRUCTION

MATCH TEXT1,P1,L1,TEXT 2,P2,L2



IF MATCH OCCURS THEN COND.REG.=0
RESULT:



4.3.5 The Match instruction (MATCH)

The MATCH instruction is used to search for the occurence of a string or partof a string within another string. The condition register will be set to zero if a match is found, or 4 if there is no match.

If a match occurs then the second operand (Start-1) will be set to the position where the match was found; if there was no match then the contents will be undefined.

The instruction format is.-

```
MATCH     String-1, Start-1, No.-1, String-2, Start-2, No.-2
```

String-1	- is the character string to be searched
Start-1	- is the position at which the search will start in String-1.
No.-1	- is the number of characters to be searched in String-1.
String-2	- is the data item containing the string to be matched.
Start-2	- is a binary data item giving the position in String-2 for the start of the string that is to be compared with String-1.
No.-2	- is a binary data item containing the number of characters in String-2 that are to be matched with String-1. This data item must contain a number that is less than or equal to No.-1.

The two pointers (Start-1 and Start-2) assume that the first character position is zero.

Example of the MATCH instruction

```
MOVE      S1,=W'0'
MOVE      S2,=W'27'
MOVE      S3,=W'3'
MATCH     VAL,S1,S2,INP,S3,S3
BE        OK
```

If VAL contained '001,002,003,004,005,006,007' and INP 'ID=005', then after the MATCH instruction has been executed control will be transferred to the statement identifier OK and S1 will be set to '16'.

INSTRUCTION	PAGE IN M04
MATCH	1.4.125

4.4 Branch instructions

An important consideration when writing any real time application, is that all potential error situations are detected and handled correctly; to aid the programmer achieve this objective many CREDIT commands use the Condition Register to record the status after execution.

CREDIT provides a wide variety of branch instructions to transfer control according to the contents of the condition register. There are also branch instructions which allow the comparison of two data items and branch if a certain condition occurs, to branch on boolean data items, indexed branches and unconditional branches.

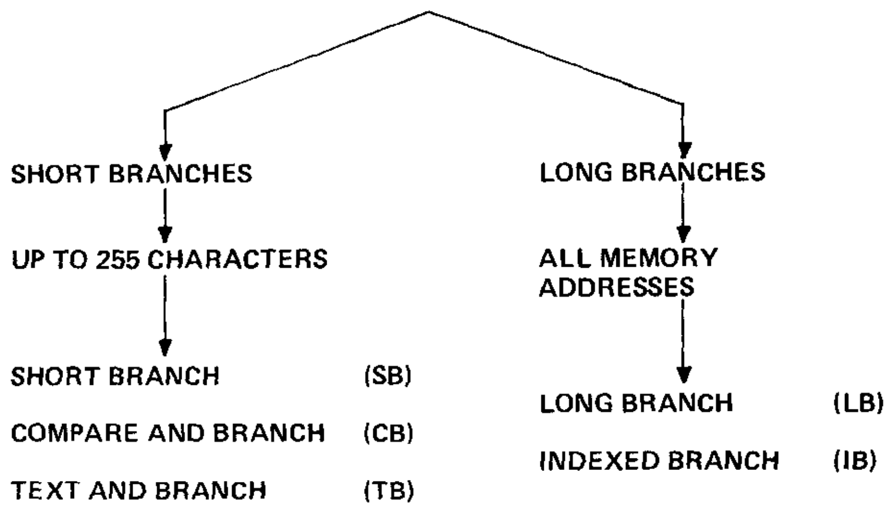
The CREDIT Translator produces two kinds of branches:

. Short branches where the destination is within 255 bytes of the branch

. Long branches for all other situations.

Short branches have a one byte displacement for holding the destination address; however a long branch has an index to T:BAT (the long branch table) where the address of the destination is held.

BRANCH INSTRUCTIONS



4.4.1 Unconditional branches

The unconditional branch instruction enables the transfer of control to the statement label identifier specified in the operand.

The instruction format is:-

```
    B     OP1
```

OP1 is the statement label identifier to which the program will branch after having encountered this instruction.

Example of an unconditional branch

```
    B     FRED1
```

The program will always branch to the statement label identifier FRED1 when this statement is encountered.

INSTRUCTION	PAGE IN M04
B	1.4.25

BRANCH WITH CONDITION MASK

```
          LB      1,        CONT1
-----------------------------------
GR        EQU     1

          LB      GR, CONT1
-----------------------------------
GR        EQU     1

          CB      GR, INLEN, CBINO, RDERR2
-----------------------------------
          IB      INDEX, SYS20, SYS40
```

4.4.2 Branch on condition mask

These instructions enable a branch to be made according to the contents of a condition mask.

The general format of this instruction is:-

```
  {SB}
  { B}   [<COND>,]<statement identifier>
  {LB}
```

If the branch instruction 'B' is used, the Translator decides whether it is a long or short branch and produces the appropriate object code. SB and LB are the mnemonics for long and short branch respectivly.

The <COND> is optional, and if present gives the appropriate condition mask for the branch, as shown in the table below; if this field is ommitted then it becomes an unconditional branch.

Cond. Code	Cause of this condition
0	Zero result from arithmetic operation Equality found with Compare instruction Logical data item had previous value of false I/O operation completed satisfactorly
1	Positive result from arithmetic operation Operand-1 greater than Operand-2 in Compare instruction End of file detected on I/O operation
2	Negative result from arithmetic operation Operand-1 less than Operand-2 in Compare instruction I/O error on an I/O operation
3	Arithmetic overflow Beginning or End of device on I/O operation
4	Inverse of condition code zero
5	Inverse of condition code one
6	Inverse of condition code two
7	Unconditional branch

Example of conditional branch

```
        B    5,L1
```

This will cause a branch to statement identifier L1 if the condition mask is equal to five (negative, or operand one less than or equal to operand two, or no end of file encountered by I/O operation)

INSTRUCTION	PAGE IN M04
B	1.4.25

4.4.3 Mnemonic branches

To make the branch instructions easier to use the code can be replaced by a mnemonic branch. The mnemonic branches use the Condition Register to establish whether or not control is to be transferred, and can be divided into three sections:

- Those for use after I/O operations

- Those for use after the CMP instruction

- Those for use after arithmetic instructions

These instructions have only one operand and this contains the statement label identifier to which control will be transferred, should the condition be satisfied.

4.4.3.1 Conditional branch after I/O instructions

Both input and output instructions cause the Condition Register to be set when they are executed. It is regarded as good program design to include checks to detect any errors that have occurred as a result of an input or output operation, and have routines to handle these situations. If an error has occurred more detail can be obtained with the XSTAT instruction, as shown in section 6.1.1.

The branch commands available for use after input or output operations are shown in the table below.

Note:

Not all these conditions can be generated by all I/O operations, for example, EOF condition will not occur when writing to a display.

The following mnemonics are used for these branches:

BEOF	Branch if End of File
BERR	Branch if Error
BEOD	Branch if End of Device
BNOK	Branch if not OK (Same as BERR)
BNEOF	Branch if not End of File
BNERR	Branch if no Error
BOK	Branch if OK (Same as BNERR)

INSTRUCTION	PAGE IN M04
BEOF	1.4.28
BERR	1.4.29
BEOD	1.4.26
BNOK	1.4.39
BNEOF	1.4.34
BNERR	1.4.35
BOK	1.4.43

4.4.3.2 Conditional branch after compare

Two operands may be compared using the CMP command described in section 4.1.6; as a result of this comparison the Condition Register is set, and can be used by branch instructions.

These branch instructions have one operand which is the statement identifier to which control will be passed if the condition defined in the branch mnemonic matches that set by the compare.

The example below shows the branch mnemonics that can be used after the compare instruction.

Example of conditional branching after compare

```
CMP     OLD,NEW                 COMPARE TWO FIELDS
BE      L1                      BRANCH IF EQUAL
BG      L2                      OLD > NEW
B       L3                      OLD < NEW
```

If OLD contained the value 5 and NEW the value 5 then control would be passed to the statement at label L1. If OLD contained the value 6 and NEW the value 5 then control would be passed to the statement at label L2, otherwise control is passed to L3 unconditionally since OLD must be less than NEW.

The conditional branches may be one of the following:

BE	Branch if equal (Operand-1 = Operand-2)
BG	Branch if greater (Operand-1 > Operand-2)
BL	Branch if less (Operand-1 < Operand-2)
BNE	Branch if not equal (Operand-1 ¬= Operand-2)
BNG	Branch if not greater (Operand-1 ¬> Operand-2)
BNL	Branch if not less (Operand-1 ¬< Operand-2)

INSTRUCTION	PAGE IN M04
BE	1.4.27
BG	1.4.30
BL	1.4.31
BNE	1.4.33
BNG	1.4.36
BNL	1.4.37

4.4.3.3 Conditional branch after arithmetic instruction

After an arithmetic instruction has been obeyed the Condition Register will contain information about the resultant value, if it was positive, zero or negative or if overflow had occurred. The branch instructions which can be used after arithmetic instructions are:-

BN	Branch if result < 0
BNN	Branch if result > 0 or = 0
BNP	Branch if result < 0 or = 0
BNZ	Branch if result < 0 or > 0
BOFL	Branch if overflow occurred
BP	Branch if result > 0
BZ	Branch if result = 0

Example of conditional branch

The following section is from a program which is being used for processing binary (BIN) data items.

```
                MUL      AMM,XRATE                   CONVERTED AMOUNT
                BOFL     OVF57                       OVERFLOW
                BNP      MC17                        NO POSITIVE AMOUNT
                MOVE     WK1,AMM                     WORKING STORE VAR.
                MUL      WK1,=W'175'
                BOFL     OVF58                       OVERFLOW
                DVR      WK1,=W'100'                 AMM*1.75
                MOVE     BAL,WK1                     STORE
                B        MC18
      MC17      MOVE     BAL,AMM
      MC18                                           RESULT NOW IN BAL
```

If AMM contained 1700 and XRATE 450, then after the multiply command had been executed, the contents of AMM would be undefined, though the overflow bit in the condition register would have been set. When the branch on overflow (BOFL) command is encountered a branch would be made to the statement identifier OVF57.

If AMM contained 0 and XRATE 450, then a branch would be made to the statement identifier MC17.

Following the multiplication of WK1 by the constant 175 overflow may occur (the largest binary number that the machine can store is 32767), and a test is made. If overflow has occurred then a branch would be made to OVF58. However, as there is no risk of overflow or division by zero at the DVR command, no checks follow that instruction.

INSTRUCTION	PAGE IN M04
BN	1.4.32
BNN	1.4.38
BNP	1.4.40
BNZ	1.4.41
BOFL	1.4.42
BP	1.4.44
BZ	1.4.45

4.4.4 Compare and branch instructions

These instructions are combined compare and branch instructions; but they can only be used where a short branch would be generated, hence the destination address of the branch must be within 255 bytes of the current address. If the destination address is greater then 255 bytes from the current address then the following section of code could be used.

```
      CMP    A,B
      BE     L1
```

However for short branch situations the following combined instruction is used.

```
      CBE    A,B,L1
```

The rules for data types in the compare instruction also apply to the data types in the compare and branch instructions. This is a three operand instruction of the format:-

```
         CB<type>      data item 1, data item 2, statement identifier
```

The first operand must be a data item identifier.

The second operand is either a data item identifier or a literal.

The third operand is the statement identifier which will be branched to if the condition specified in the compare and branch instruction is satisfied by the first two operands.

The type is one of E,G,L,NE,NG,NL and is used to form the mnemonics listed below.

Examples of the compare and branch instruction

```
    CBL       BAL,AMM,OVD           Branch to OVD if BAL is less than AMM
    CBE       AMM,MAXL,SPC1         Branch to SPC1 if AMM is equal to MAXL
    CBNE      LOOP,=W'1',ROUND      Branch to ROUND if LOOP is not equal to 1
    CBL       RATE,IRR,GO           Branch to GO if RATE is less than IRR
    CBG       AMT,FIX,ERR           Branch to ERR if AMT is greater than FIX
    CBNG      ACC,MIN,READ          Branch to READ if ACC is not greater than MIN
```

INSTRUCTION	PAGE IN M04
CBE	1.4.49
CBG	1.4.51
CBL	1.4.53
CBNE	1.4.55
CBNG	1.4.57
CBNL	1.4.59

4.4.5 Test and branch

These instructions can only be used for branching dependent on the condition of boolean data items, and have the following format:-

```
        command         boolean variable, statement identifier
```

The command can either be test and branch if true (TBT) or, test and branch if false (TBF).

The first operand specifies the boolean data item on which the decision to branch or not will be made.

The second operand is the statement identifier to which control will be passed should operand one satisfy the criteria of the command.

Example of the test and branch command

```
    TBT        STAT,MC17
```

This will cause a branch to statement label MC17, if STAT holds the value TRUE otherwise the next consecutive instruction will be obeyed.

INSTRUCTION	PAGE IN M04
TBF	1.4.52
TBT	1.4.53

4.4.6 Indexed branch instructions

The indexed branch command (IB) is used to generate a long or short branch to one of a number of statement labels depending upon the value of an index.

The format of the indexed branch command is :-

```
    IB        Index, Label-1, Label-2,..... Label-n
```

Index - is a binary data item containing the index to be used by the branch.

Label-1 etc. - these are a list of statement identifiers to which the program may branch depending upon the value in the index. If the index held the value one then this would cause a branch to the statement at Label-1, if the index held the value 2 then a branch would be made to the statement at Label-2 and so on.

If the index contains the value zero, or a value greater than the number of statement label supplied, then the next consecutive instruction will be executed.

Example of an indexed branch

```
    IB        SPBINW2,READIN,DUMMEY,KEOI,KTFWD,KTBWD,KTHOME,              C
              KTLDOWN,KTLEFT,KTRIGHT,KTUP,KENTER
    SUB       SPBINW2,=W'14'
```

SPBINW2 is the binary index used for controlling this indexed branch; where the program branches to depends on the contents of this data item.

Contents of SPBINW2	Statement identifier to which control will be passed
1	READIN
2	DUMMEY
3	KEOI
4	KTFWD
5	KTBWD
6	KTHOME
7	KTLDOWN
8	KTLEFT
9	KTRIGHT
10	KTDOWN
11	KTUP
12	KENTER

If SPBINW2 contains the value zero, or a value greater then twelve then the next consecutive statement will be executed, in this case the subtract command.

INSTRUCTION	PAGE IN M04
IB	1.4.114