10.12 Input and Output using Private Magnetic Tapes

All the documents fed to peripherals for input are stored by the computer on a magnetic tape known as the system input tape. They are then brought into the core store as required. If a program has a large amount of input it may be in the programmer's interest, and in the interest of the efficiency of the computer, to transfer this input to a private tape before starting his program, instead of using the system input tape.

Output is normally accumulated on the system output tape before being sent to the required peripherals. If a program involves extensive output this may be written to a private magnetic tape instead, and later output upon requesting the Supervisor to do so by means of a special document fed in through one of the peripherals after the job is complete.

Both these facilities are dealt with in this section.

10.12.1 Extensive Input

A document can be copied to a NEW magnetic tape by putting above the document heading the directive:-

```
COPY TAPE NEW
*tape number  tape title
```

The title specified will be written in section 0 of the new tape and the document following will be copied into section 1 onwards.

If it is required to store the document on a previously used tape (i.e. one which already has a title) the necessary heading is:-

```
COPY TAPE b
*tape number  tape title
```

Here b is the number of the tape block at which it is intended to begin copying the input.

When the copying process is complete the following information will be printed by the computer:-

```
*tape serial number/section/word
 title document.
```

The section and word indicate where the document is stored.

Example:
To copy the first data document of the job given in 10.5 to the NEW tape F1111 it would be fed to the peripheral in the form:-

```
COPY TAPE NEW
*F1111 DATA FOR F64, METALS
DATA
F64/A, IRON CONTENT
- - - - - - - - - - - -
(data)
- - - - - - - - - - - -
- - - - - - - - - - - -
***Z
```

The information output by the computer would be:-

```
*F1111/1/0
F64/A,  IRON CONTENT
```

The second data document could be sent to section 7 onward of the same tape thus:-

```
COPY TAPE  7
*F1111 DATA FOR F64, METALS
DATA
F64/B, COPPER CONTENT
- - - - - - - - - - - -
(data)
- - - - - - - - - - - -
- - - - - - - - - - - -
***Z
```

This could produce the output:-

```
*F1111/7/0
F64/B, COPPER CONTENT
```

10.12.2 Job Description References

In order to use documents copied to private tape, the tape must appear in the tape section of the job description in the usual way. In the INPUT section of the job description the sub-heading:-

```
i TAPE a/b/c
```

must appear before the title of the document. Here i is the input stream number, a is the programmer's number for the tape, b the block number, and c the word number within block b at which the document starts. Thus the job description to run the program of 10.5 could begin:

```
JOB
F64, STATISTICAL ANALYSIS OF METALS
INPUT
0 F64, ANALYSIS PROGRAM
1 TAPE 27/1/0
F64/A, IRON CONTENT
2 TAPE 27/7/0
F64/B, COPPER CONTENT
OUTPUT
0 LINE PRINTER
1 FIVE HOLE PUNCH 3 BLOCKS
TAPE
27 *F1111 DATA FOR F64, METALS
```

10.12.3 Re-use of Documents on System Tapes

As already explained, under normal circumstances documents are stored on system tapes before use. Among the information on output 0 will be the location of each document on tape. This will take the form:-

```
Sa/b/c
```

where Sa is the system tape number, and b,c are the block and word numbers of the first word of the document. Such tapes will be stored for a fixed period of time and if a document is required again within this time it may be called for direct from system tape, avoiding the use of a slow peripheral. This is done by including in the INPUT section

```
        i  TAPE  Sa/b/c
        Title of document
```

where i is the programmer's number for the document.

For example, if a program called 'F74 FACTORISATION' has been run and is stored at S7/2/411 it can be re-run with a data document called 'F74 LIST 3A' by a job description as follows:-

```
        JOB
        F74 FACTORISATION RUN 2
        INPUT
        0 TAPE S7/2/411
        F74 FACTORISATION
        1 F74 LIST 3A
        OUTPUT
        1 LINE PRINTER
        ***Z
```

Only the job description and the data document would then need to be fed to peripherals. Note that no further reference to the system tape is required.

10.12.4. Extensive Output

Large quantities of output may be written to a NEW magnetic tape by specifying in the OUTPUT section:-

```
        OUTPUT
        i TAPE a/b/c
        type of equipment m BLOCKS
```

Here, i is the output stream number, a the programmer's number for the tape, b the block number, and c the word number where the copying is to start. The last line is as in section 10.4.2.

Thus if output 1 of the job in section 10.5 were 300 blocks instead of 3 the OUTPUT section could be written:-

```
        OUTPUT
        0 LINE PRINTER
        1 TAPE 27/1/0
        SEVEN HOLE PUNCH 300 BLOCKS
```

The tape must be specified in the usual way in the tape section of the job description:-

```
        TAPE NEW
        27 *F1111 F64, OUTPUT METALS
```

This will cause the NEW tape F1111 to be given the title 'F64, OUTPUT METALS' and the output to be sent to this tape beginning at block 1.

If it is required to store the output on a previously used tape, the necessary entry in the OUTPUT section is the same, but the tape must be specified under the heading TAPE in the tape section.

The private tape can be printed by a steering tape consisting of:-

```
        PRINT TAPE
        *tape serial number/tape title
```

if the whole tape is to be printed

or

```
        PRINT TAPE b/c
        *tape serial number/tape title
```

if one document is to be printed, from block b word c onwards.

10.13 Job Description Parameter

In a job where different parts of a standard program are required to process different types of data, it may be more convenient if the data type is indicated in the job description rather than in the data or program. The parameter section in the job description provides for this, and is written in the form

```
                    PARAMETER
                    *<number>
```

The number consists of up to eight octal digits, and is **left justified.** PARAMETER and the number may be on the same line, separated by one or more spaces.

Examples:

```
            PARAMETER  *00061247

            PARAMETER  )
            *1060      }   alternative
                       )      forms
            PARAMETER  *1060000 )
```

When the parameter section appears with other sections, the order of sections is immaterial; if omitted, the parameter is taken to be zero.

The value of the parameter may be read by program using the instruction

```
        1140    4    0    S
```

which will set the half-word s′ = parameter number (see section 12.9).

Chapter 11

MONITORING AND FAULT DETECTION

11.1 Supervisor Monitoring

With the aid of special hardware, the Supervisor keeps a record of the progress of a program during its run. This supervisor monitoring notes amongst other things the store in use, the computing time taken, and the occurrence of errors which may prevent the successful execution of the program.

In Atlas, provision is made for the automatic detection of a variety of clearly defined fault conditions, which may be due either to mistakes in the program itself, or to errors occurring in the computer or the peripheral equipment. The faults result in entry to a part of the Supervisor known as the 'monitor' program, with the particular fault responsible being distinguished by a mark or count set in B91.

The monitor program consists of a set of routines, some in the fixed store and some in the main store, whose primary purpose is to print out such information as will enable the cause of the fault to be diagnosed. In the case of certain types of program fault, it may be possible for the programmer to decide beforehand what action should be taken to enable the program to be resumed; he will then provide a number of fault routines and list them in the trapping vector. The monitor program will enter the trap indicated in B91, if such indication does in fact correspond with an entry in the trapping vector; otherwise it will proceed to diagnostic printing, or will transfer control to a private monitor routine if so requested. This private monitor program may provide a special form of diagnostic printing - either instead of or in addition to the standard - and may, once and once only, cause the program to be resumed. Any subsequent monitoring will always be followed by the End Program sequence, except when the fault is trapped.

When entry to the private monitor follows expiry of the total time allotted to the program, a further 4 seconds of computing time is allowed for the completion of the private monitor routine. Similarly, if execution time is exceeded, a further minute of execution time is allowed, and if Output is exceeded, a further block is allowed.

11.1.1 Types of Program Faults

We shall here concern outselves solely with program faults: there is little that the ordinary programmer can do about machine faults, other than to minimize their effect by providing adequate re-entry points and an informative private monitor routine.

There are three distinguishable categories of fault detection causing entry to the monitor program:-

(a) '*Interrupt Faults*' Some faults are detected by special equipment provided for the purpose: these include exponent overflow, division overflow, use of an unassigned function code, and 'sacred violation' (i.e. reference to a part of the store reserved by the Supervisor). An interrupt occurs and will set in B91 a digit corresponding to each such fault. Further analysis of the fault is provided by a supervisor extracode routine (S.E.R.), the same routine being used for all faults.

(b) '*Supervisor Faults*.' A further class of faults are detected by S.E.R.'s, being especially concerned with faulty use of the store and of peripheral equipments, and with the over-running of time allowances. These faults lead to the setting of an appropriate digit of B91, just as if they had been detected by hardware, as in (a) above; they include over-running of computing time or of tape waiting time, and attempts to exceed the requested store allocation. In the object program, extracode instructions dealing with the store and peripherals will enter a S.E.R. to detect faulty usage. Only one such fault can be detected at once, but it will be recorded as a count in B91 without interfering with any existing record of faults of the interrupt type. The same S.E.R. monitor sequence is entered as in (a) above.

(c) '*Extracode Faults*.' Many faults are detected by the ordinary extracode routines themselves; typical of these are errors in the arguments of functions. Only one such fault can be detected at once; the extracode will set a counter in B91 and then jump directly to the common S.E.R. monitor sequence mentioned above.

The various faults which result in entry to the monitor program are listed in the table below:-

FAULT TABLE

FAULT	MONITOR PRINTING	DETECTED BY	MARK OR COUNT IN B91	TRAP NUMBER (IF ANY)
Local time expired	L TIME EXCEEDED	S	Bit 5	0
Division Overflow	DIV OVERFLOW	I	Bit 6	1
Exponent Overflow	EXP OVERFLOW	I	Bit 14	2
Page locked down	PAGE LOCKED DOWN	S	Bit 1	3
Number of blocks exceeded	EXCESS BLOCKS	S	2.0	4
Square root argument < 0	SQRT -VE ARG	E	2.4	5
Logarithm argument ≤ 0	LOG -VE ARG	E	3.0	6
SPARE				7
Inverse trig. function	INVERSE TRIG OUT OF RANGE	E	4.0	8
Reading after Input Ended	INPUT ENDED	S	4.4	9
End of Magnetic tape	END TAPE	S	5.0	10
Variable length record error	V TAPE ERROR	E	5.4	11
Magnetic Tape failures	TAPE FAIL	S	6.0	12
Computer failures	COMPUTER FAIL	S	6.4	13
Unassigned Function	ILLEGAL FUNCTION	I	Bit 4	
Sacred Violation Instruction	SV INSTRUCTION	I	Bit 8	
Sacred Violation Operand	SV OPERAND	I	Bit 10	
Illegal block number	ILLEGAL BLOCK	S	9.6	
Band not reserved	BAND NOT RESERVED	S	10.2	
Computing time expired	C TIME EXCEEDED	S	Bit 2	
Execution time expired	E TIME EXCEEDED	S	Bit 3	
Input not defined	INPUT NOT DEFINED	S	11.6	
Output not defined	OUTPUT NOT DEFINED	S	12.2	
Output exceeded	OUTPUT EXCEEDED	S	12.6	
Tape not defined	TAPE NOT DEFINED	S	13.2	
Illegal search	ILLEGAL SEARCH	S	13.6	
No selected tape	NO TAPE SELECTED	S	14.2	
No mode defined or attempt to write when not permitted	WRONG TAPE MODE	S	14.6	
Number of decks exceeded	EXCESS DECKS	S	15.2	
No trap set	TRAP UNSET	S	15.6	
Number of branches exceeded	EXCESS BRANCHES	S	16.2	

(I = Interrupt; E = Extracode; S = S.E.R.)

Some of these faults are associated with information in the job description, and are listed below.

Excess Blocks
Computing Time exceeded
Execution Time exceeded
Input not defined
Output not defined
Output exceeded

Tape not defined

Excess Decks

Division overflow occurs with certain division instructions, marked DO, when the divisor is zero or substandard.

Exponent overflow occurs with certain instructions, marked E, when, after completing all functions of the instruction, including rounding and standardisation, the accumulator exponent is greater than +127. The guard and sign bits of b124 will be 0 and 1 respectively. If exponent overflow is trapped, the fault indication will not be reset and may cause further monitoring with a subsequent accumulator operation. This may be avoided by first clearing B124, setting guard and sign bits to zero. If the accumulator exponent becomes less than -128 during multiplication, division or standardisation, then the guard and sign bits of b124 are 1 and 0 respectively. This is exponent underflow. The accumulator is set to floating point zero, but no fault is indicated.

The arguments for square root, logarithmic and inverse trigonometric functions are all tested to determine whether they are within range. They may cause exponent overflow if very large arguments are used.

Programs that employ line reconstruction, such as the ABL compiler and L100, will attempt to read after the input stream is finished if the end of document marker ***Z etc. occurs on the same line as the last information for the program.

Sacred Violation Instruction refers to a transfer of control to address J5 or above. If control is transferred to the fixed store, between J4 and J5, there is no immediate interrupt, but the result is unpredictable.

Sacred Violation Operand always means an attempt to read or write to the private store (i.e. to J5 or above).

Illegal block number indicates a reference to the Supervisor working store, between J34 and J4.

Certain functions are detected as being illegal. The instruction

```
1000   0   0   0
```

which is the same as floating point zero, causes entry to extracode control to attempt to obey the instruction

```
001   0   0   0
```

or the floating point number $\frac{1}{2}$. This function is recognised as being unassigned.

End of magnetic tape refers to an attempt to obey a transfer instruction involving section 0, or to use tape beyond the last addressed section (section 4999 on a fully addressed 3600 ft. tape).

Illegal search refers to an attempt to search a magnetic tape for section 0 or beyond the last addressed section.

No magnetic tape selected indicates that none of the 'select tape' extracodes has been obeyed.

Variable length record error implies an attempt to transfer when not aligned on a tape marker, or an encounter with an incorrectly made marker.

Should a magnetic tape failure occur, the program may not be monitored immediately, and in this case an attempt will be made, by the Supervisor, to produce a correct transfer. After a set number of failures, usually seven, full monitoring will then occur. An attempt to read from a section that has not previously been written to will be recognised as a tape failure, and hence it is advisable to write to all sections required when a new tape is used. A decimal number may follow the TAPE FAIL text, and this is interpreted as

$$8192A + 512B + C$$

where C is the tape number on which the failure occurred

B is the fault number

and A is the fault type, 0 if E-type, 1 if F-type.

These fault types are described in the Atlas 1 Computer System Operators' Manual.

A monitor for 'Trap Unset' indicates an attempt to enter a trap using extracode 1134 when no tape has been set by extracode 1132 (see section 11.2).

An attempt to exceed the number of active branches specified by extracode 1103 will be monitored (see Chapter 12).

'Page Locked Down' and 'Band Not Defined' monitoring may occur when a program is controlling transfers between the drum and core store (see Chapter 12).

11.1.2 Standard Post Mortem

Following the detection of one or more of the faults listed above, the appropriate text will normally be printed on Output 0, followed by a standard form of post mortem printing, similar to that shown below.

```
BAND NOT RESERVED
INSTR  73          121,127,0  , 549       B127 =  0
INSTR  74         1177,0  ,12 , 0         B12  =  4.6
INSTR  75          101,13 ,12 , 108.4     B13  = -1048574.6 B12 = 4.6
B1  =-699050.6    B2  =-1031932   B3  = 42798.7     B4  =-42799
B5  =-0.1         B6  =-258553    B10 = 18          B12 = 4.6
B13 =-1048574.6   B70 = 0.6       B80 = 0.1         B90 = 36
B91 = 10.3        B92 = 0.1       B93 = 0.1         B94 = 7.1
B95 = 263.1       B96 =-0.3       B97 = 917504.4

   TAPE 1         AT   64
   TAPE 2         AT   72/306
   TAPE 3         AT   1
```

Normally the current address in main control, B127, is printed, followed by its full word contents in instruction form, and the contents of any

B-lines in the range B1 to B99 referred to, unless zero. Irrespective of their true value, the contents of any B-lines in the range B100-B127 referred to will be printed as zero. The two previous words are similarly printed. If one of the locations is in the private store or is undefined, then

INSTR UNALLOCATED

will be printed instead of the normal form. Following an EXCESS BLOCKS fault, b127 will be reduced by 1 before printing instructions. The instructions printed may not be the last three obeyed, as a jump may have occurred to either the second or third instruction printed, as must have happened in the example. Also, because of the overlap in executing instructions with each other, and with other operations such as tape transfers, the printing may occasionally bear little relation to the instruction originally causing the fault, although many extracodes causing faults will appear as the second instruction; basic functions causing faults may appear as any one of the three instructions, or not at all.

After the three instructions, the contents of all B-lines B1-B99 are printed, unless zero.

If one inch magnetic tapes are being used, their current positions are then printed out, indicating the next section number, and, for variable length working, the word number also.

Except for the function codes, which have their usual form, and octal fractions in addresses and B-lines, all numbers are decimal.

11.1.3 Ending a Program

If no fault is found during the execution of a program, then the run will be finished by obeying the 1117 extracode. If a fault is found, then, after post mortem printing, the Supervisor will normally end the run as if the 1117 instruction had been obeyed.

1117 Print monitoring information on output 0. End all program output streams, indicating the amount of output in each stream. Instruct the operators to disengage and rewind all magnetic tapes used. Remove the job from the store; clear Supervisor directories relating to this job.

The form of the monitoring information on output 0 is given below.

```
<Job Title>
INSTRUCTION 8      7
STORE 32        / 4
 2 DECKS 6      TAPE BLOCKS 1      HALT TIME
INPUT 0        1        BLOCKS
OUTPUT 0    (3) ANY 40       RECORDS
```

The meaning of this information is explained below.

```
INSTRUCTION 8      7
```

8 instruction interrupts were obeyed in all, of which 7 were used in compiling. An instruction interrupt occurs every 2048 instructions, basic multiplication and division orders being counted as 2 and 4 instructions res-

pectively. At some installations a third number on this line gives the number of program blocks transferred between drum and core store.

```
STORE 32        / 4
```

The job description requested 32 blocks of store, of which 4 blocks were in use when the program was terminated, including one block for each input or output stream.

```
2 DECKS 6       TAPE BLOCKS 1      HALT TIME
```

Two magnetic tape decks were reserved, and six blocks were transferred between tape and main store. The program was suspended by the Supervisor for one second awaiting the completion of tape transfers.

```
INPUT 0           1        BLOCKS
```

One block was needed in the input well to hold the information on input stream 0. There will be similar printing for all other input streams defined.

```
OUTPUT 0      (3) ANY 40      RECORDS
```

The job description requested that output stream 0 should be to ANY type of peripheral. Forty records were output before the program was terminated. If the actual peripheral is a paper tape punch, the output will be measured in blocks; otherwise, in records. The output stream was broken into three parts; this printing is suppressed if the stream was not broken. There will be similar printing for all other output streams defined.

The amount of output, indicated at the end of each stream, is always measured in blocks, in the form

```
END OUTPUT        1 BLOCKS
```

11.2 The Trapping Vector

Upon entry following a fault, the first action taken by the monitor program is a check to see if the fault has been trapped by the programmer; if so, the monitor sequence will at once exit to the trap set.

The trapping vector occupies several successive words of the store, and the address of the first word must be specified as S in the extracode 1132. Each word holds the trapping information for a particular fault, word n being associated with fault type n; the more-significant half-word contains the address of the fault routine to which the trap will transfer control, and the less-significant half-word contains, in bits 15-21, the address of a B register which is used to hold the value of main control when the fault was detected. This may be but is not necessarily the point in the main program at which the fault occurred.

Only some of the faults listed in the table in section 11.1.1 have trap numbers: these are the faults which the programmer might reasonably be expected to deal with before resuming the program; certain traps may be useful as a means of avoiding extra testing in the program. There are faults which are not trappable; these include such faults as sacred violation, which the programmer can be expected to avoid, and deviations from the job description, in exceeding the specified time allowance, for instance.

No trapping will occur unless the program first obeys an 1132 instruction, specifying the address of a trap vector. Subsequently, trapping can be suspended by specifying a negative address in extracode 1132. In order to trap some faults but not others, the programmer should specify a negative jump address in each unwanted trap. Normally the trap vector will be punched as part of the program, and the unwanted entries may be punched as +0 or merely omitted, since floating-point zero produces a negative first half-word, because its exponent is -128.

Two other extracodes associated with the trapping vector are given below.

1133 Place the first address of the trapping vector, if any, in Ba; if no trap has been set, make ba negative. This enables a subroutine to preserve the current trap when a private trapping vector is required.

1134 Obey the entry number Ba in the trapping vector, as though a fault of type Ba had occurred. Ba may range from 0 to 63 inclusive, and may be used to enter standard traps, or as a subroutine entry at addresses listed in the trapping vector for 'fault' numbers 14 onwards.

11.3 Private Monitoring

When the monitor program encounters a fault which is not trapped, it prepares to terminate the program and proceed to diagnostic printing, as described earlier. If he so desires, the programmer may supply a private monitor sequence, whose starting address must be specified by using extracode 1112. The last octal digit of the starting address determines the time of entry to the private monitor as follows:-

Octal Fraction	Entry
1	Before printing the one line explanatory text
0	After printing the text
2	After the standard post mortem printing

When the entry is before any printing, B91 contains the record of faults, and B92 the value of main control when the fault was detected, with the contents of B93 and B121 altered. Otherwise B96 and B97 will also be altered.

In the event of faults in the private monitor sequence itself, it is necessary to avoid the possibility of endless loops of errors; this is accomplished by forbidding a second entry to the private monitor. Any subsequent faults may be trapped, but if they are not trapped the standard monitor will end the program.

Specifying a negative address with extracode 1112 cancels any private monitor routine.

11.4 Restarting and Re-entering a program

Following a failure in the computer or an on-line peripheral, jobs completely in the machine will not be lost, although documents of incomplete jobs, and documents only partially read must be input again. Programs partially executed will normally be restarted from the beginning; there are no facilities at present by which the Supervisor will dump program information to allow re-entry to a point other than the start of a program.

11.4.1 Preventing a Restart

It may be useful to prevent a job using the 'break output' facility from being restarted once a point is reached where the job is substantially complete; alternatively a restart may be impracticable for a job using magnetic tape. To this end, the instruction

```
1113    0    0    -1
```

will prevent the job being restarted if a breakdown occurs following the use of the extracode, but before completion of execution.

11.4.2 Re-entering a Program

After a breakdown, for a program to be re-entered at some point other than its start, it is normally necessary to dump information as the job proceeds, specifying a re-entry point with each dump. Although the Supervisor does not yet provide such facilities, the programmer may provide his own dump routine.

Such a routine, called Dumpling, is described in the I.C.T. Atlas Computing Service Bulletin No. 7. Dumpling occupies one main store block, dumping, on to magnetic tape, the information listed below:-

All defined store blocks, including Dumpling.

The tape numbers and positions of all one inch magnetic tapes working in fixed length mode.

The accumulator (double length)

The logical accumulator

B-lines 1 to 90, and B121

The number of the currently selected input stream.

The number of the currently selected output stream.

V-store line 6 containing A0, Bt, Bc, etc.

The address of the trapping vector, if any.

Details of the dump region.

The dump number.

The re-entry point to the program in the event of a breakdown.

As each dump is made, its location is printed on output stream 0.

After a breakdown, a very short steering program allows the information stored at the last or the penultimate dump to be recovered, and the program continues from the corresponding re-entry point.

11.5 Monitor Extracodes

The monitor extracodes introduced earlier are listed again here.

1112 Set the address of the private monitor routine to S. If the standard monitor program is subsequently entered, following a fault which is not trapped, control will be transferred to the private monitor, possibly after some diagnostic printing. The amount of diagnostic printing is determined by the octal fraction of S, as follows:

Octal fraction	Print-out
0	One line describing the fault.
1	No standard printing.
2	One line describing the fault, followed by standard post mortem print-out.

To subsequently suspend private monitoring, a negative address, S, must be used with 1112.

1113 0 0 -1 Do not restart.
If a breakdown occurs after obeying this instruction but before the job is completed, it will not be restarted.

1117 End program
Print monitoring information on output 0; end all output streams. Instruct the operators to disengage and rewind any magnetic tapes used. Clear all Supervisor references to this job.

1132 Set the address of the trapping vector to S.
This extracode must be obeyed before any trapping can take place; to subsequently suspend trapping, 1132 must be used again, but this time with S negative.

1133 Find address of trapping vector.
Set ba' to the first address of the trapping vector if one is defined, otherwise set ba' negative.

1134 Enter trap Ba. ($0 \leq Ba \leq 63$)
Obey entry number Ba in the trapping vector, as though a fault of type Ba had occurred.

11.6 Faults Detected by the Compiler

Apart from faults detected during the execution of the program, many types of error may be found earlier by the ABL compiler. An indication of the type and location of each fault is printed out on the current output stream, usually output 0, and, if necessary, arbitrary values are assigned to expressions to allow compiling to continue.

There are some special preset parameters which determine the exact action taken by the compiler after a fault has been found. These parameters are described in Chapter 12; the compiler action described in this section assumes no program setting of these parameters. In particular, P110 will normally be zero.

11.6.1 B-lines in ABL

It may be useful to know whether a run has ended during compiling or execution. When ABL is in operation, B3 is in the range -127 to 0 inclusive. The most likely value is -127, unless the run has been terminated by INPUT ENDED.

ABL uses most of the B-lines. It preserves its own B-lines when it meets an enter directive, and then clears B1 - B88 before obeying the directive. B89 will contain the current value of *. After an E-directive, B90 contains J3; after ER or EX, B90 is clear.

11.6.2 Indeterminate Items

When ABL needs to evaluate an expression, and can not do so, the expression is faulted as described below, and an arbitrary value assigned. The value depends on the context.

(i) * = expression. When this is faulted, * is given the value 654321P110 or 654321.1P110. This allows ABL to try to check the rest of the program, although it may not be able to enter it.

(ii) In all other cases the expression is given the value J36. If the expression is in an Enter directive, it will cause a monitor on ILLEGAL BLOCK, since J36 is an address in the Supervisor Working Store. A similar monitor is likely if the expression occurs in an instruction which is subsequently obeyed.

When any other faulty item is found, nothing is compiled. As the store is initially cleared to floating point zero, all or part of this bit pattern will remain in the location reserved for the faulty item.

11.6.3 Diagnostic Printing

The first ABL diagnostic printing for a program is preceded by the line of printing,

ABL MONITORING

Each fault causes printing, on a new line, of the location of the fault within the program, together with an explanatory text. The following line will usually be a reconstruction of the line of program containing the fault.

If the error density is higher than 8 faults in 24 lines of print-out, then

TOO MANY ERRORS

is printed, and the run ended (see also P101). Blank lines, and lines containing only erases, are ignored.

Normally, when a fault is found, compiling continues until an E or ER type of enter directive is encountered, so that any further faults may be detected. When the enter directive is reached after one or more faults,

ERRORS DO NOT ENTER

is printed, and the run ended (see also P102).

11.6.4 Fault Location

Each ABL monitor printing begins with a specification of 'where' in the program the offending item occurs. 'Where' means 'in what line of the printed program and in what part of that line'. ABL refers to lines of print in exactly the same way as a programmer does by counting from the last label set, but with the exception that A0 is not used to mean the line on which the first instruction or item of a routine appears but the line in which the R itself appears. Thus the very first line of print in a program is 1 A0/0. Blank lines, and lines containing only erases, are ignored.

When ABL prints 3,4 A1/20 it means that it has read 3 terminators in line 4 after A1 of routine 20 when it has found an error. Thus, for example, if an expression is incomplete (e.g., no close bracket after an open bracket), then the next terminator will have been encountered before ABL can realise that there is an error, and the 'position along the line' will be printed accordingly; but, if a bogus character is found in an expression, then this will be recognized before the next terminator is encountered.

11.6.5 Diagnostic Printing Character Set

In all diagnostic printing, ABL uses its own character set. The following is a list of all available external characters (7-track tape, 5-track tape, and cards) and their corresponding ABL characters. All the ABL characters are contained in the 7-track tape and Anelex line-printer character sets, so that ABL diagnostic printing may always be completely printed or punched on these media.

External:	0	1	2	3	4	5	6	7	8	9
ABL :	0	1	2	3	4	5	6	7	8	9

External:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
ABL :	A	B	C	D	E	F	G	H	1	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

External:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
ABL :	A	B	C	D	E	F	G	H	1	J	K	∝	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

External:	½	/	∝	β	.	+	-	:	'	[	]	<	>	=	_	,	(	)	?	²	π	&	|
ABL :	β	/	∝	β	.	+	-	:	'	[	]	<	>	=	∝	;	(	)	?	∝	|	M	|

External:	*	10	11	%	£	≠	∨	≈	≥	→	x	∅	n
ABL :	*	∝	∝	∝	|	∝	∝	∝	∝	∝	∝	∝	'

All other single characters are replaced by ∝. All composite characters not including Erase are replaced by β. All composite characters including Erase are ignored. All single spaces are ignored. Two consecutive spaces are replaced by Comma. After a Terminator all spaces are ignored, but not commas (see fault below).

l (lower case L) is an illegal character, unlike the rest of the lower case alphabet; i and o are treated as one and zero respectively, as are capital I and O.

11.6.6 Explanatory Texts

The ABL texts are listed below, with further explanation where necessary.

INSTRUCTION?
ABL thinks that a line contains an instruction, but something is incorrect with the format, i.e. a line begins with a number (unsigned), and does not have four parts.

OCTAL NO. CONTAINS 8/9

IRREGULAR FUNCTION
ABL thinks that a line contains an instruction, and its first part has not been faulted by either of the two preceding monitors, but contains either more than 4 digits, or four with the first equal to 2 or more.

WRONG FORMAT
A line contains items which cannot be identified; whatever kind of items they are meant to be, something is wrong.

NOT TERMINATED
A character which has no meaning in the current context is encountered within an item. When this occurs, ABL skips to the next terminator.

ACCUMULATOR OVERFLOW
Fixed-Point Overflow is caused in the Accumulator as a result of any arithmetic process required because of the form of any item. In practice, this can only occur during the 'de-standardising' process required by 'd' in the Floating-Point Number forms a:d, a(b):d, a(b:c):d, and a(:c):d.

SHIFT >23 PLACES   Requested by a U or a D operator.

IMPERMISSIBLE +

Z in RO

* OUT OF RANGE
An attempt is made to compile an item into store with the transfer address (*-P110) greater than or equal to J3, or less than 0. Compiling will continue from *=654321.1 + P110 in an attempt to detect any other faults.

PARAMETER OR ROUTINE NO. TOO BIG
A parameter, routine, or copy number is greater than the permitted highest value, that is

(i) if a reference is made to a routine number greater than 3999

(ii) if a routine parameter Aa is encountered with a >3999

(iii) if a global parameter Ga is encountered with a >3999

(iv) if an attempt is made to set a preset parameter in the range P110 - P119, unless it is one of the Special Preset Parameters, to set a preset parameter Pa with a >119, or to use a preset parameter Pa, with a >129

(v) if a reference is made to a library routine number greater than 1999

(vi) if a reference is made to a copy number of a library routine greater than 1999

NOT IN LIBRARY
One of the forms L, La, La.b, E, ER, RLc appears within a library routine. The monitoring occurs as the library routine is compiled, but this monitor should only affect library routine writers or private library routine users.

EXCESS COMMA
Two or more commas or a comma after Multiple Space appear between items or parts of an instruction on a line. A comma at the beginning of a line will give rise to a WRONG FORMAT monitor. For the purpose of this fault, comma is not the same as multiple space; thus, for example, several TABS are perfectly permissible between items.

LABEL NOT ALLOWED

For example, before an R or a T directive.

? This occurs whenever the Supervisor, as opposed to the ABL compiler, monitors any aspect of the program. The most common cases are:

Exponent Overflow	- (in any arithmetic arising from the form of any items)
Input Not Defined	- (after a use of the 'P115=expression' directive)
Output Not Defined	- (after the use of the 'Ta' or 'Ta-b' directive)
Input Ended.	

The normal Supervisor fault printing comes first followed by an ABL '?' monitor giving 'where' and the reconstructed line in the usual form.

This kind of monitoring can only occur once, because of the Supervisor's system of private monitoring. If a second error is caught by the Supervisor, only the Supervisor printing will appear and the job will be terminated.

If this is the first fault in the program, then the line 'ABL MONITORING' will appear after the Supervisor monitor printing but before the '?' monitor printing.

The monitor Input Ended may be due to a variety of causes. The most common of these are:

(i) Unmatched [. In this case the compiler will scan through the whole of the program looking for a matching] until it hits Input Ended. This circumstance can be most easily identified by the Line Count part of 'where' in the accompanying ABL monitoring. The compiler does not recognize labels whilst within a [] sequence, but does count lines in the normal manner, so the line count will be from the last label before the [.

(ii) Un-terminated Enter Directive. This is tricky to spot because the reconstructed line looks normal. In order to avoid this monitor the ***Z or other document terminator must not be on the same line as the final Enter directive, since ABL inputs and reconstructs the whole record before attempting to identify items.

(iii) Enter Directive omitted completely.

(iv) Un-terminated private Library Routine (i.e. the 'ZL' record omitted or punched incorrectly). In this case the whole of the remainder of the program is thought by ABL to be part of the Library Routine.

<Parameter > ALREADY SET AT <where>

An attempt is made to set a parameter (Global or Routine) which has already been set. The monitor here consists of one line only; no 'reconstructed line' is printed. An 'RO' directive will also cause this fault. The parameter will retain its original value.

EXPRESSION INDETERMINATE

An expression, which has not been rejected for any of the above reasons, cannot be fully evaluated when it should be. For example, if it is the right-hand side of a 'P=expression' or '*=expression' directive or an Enter directive, then it needs to be evaluated immediately, and if it cannot be (for example, because of unset parameters), then this monitoring will occur when the item is encountered - in the case of E or ER directives after any required Library Routines have been compiled; or, if it is the address part of an instruction, or a Half-word or Six-bit word, then if it is still not determinate when the next E or ER directive is encountered - after any required library routines have been compiled - then this monitoring will occur when the Enter Directive is encountered, after any monitoring arising from any library routines. For the purpose of counting errors, all parameters unset at the Enter directive are treated as one fault.

The second line of this monitor printing consists not of the complete reconstructed line in which the offending expression occurred, but only of the expression itself, and this is not normally in its original form but has been partly evaluated, including the replacement of all set parameters by their values.

Because of this system of printing the expression, with known parameter values fully substituted, the fact that, for example, the parameter A4/2 appears in the monitor printing indicates that it is that parameter which is unset and is causing the expression to be indeterminate.

A reference in an expression to a parameter of a non-existent library routine will give rise to EXPRESSION INDETERMINATE monitor printing when the next E or ER directive is encountered as well as a monitor printing for 'Library Routine NONEXISTENT' (see Chapter 12).

An attempt to divide by zero in an expression gives rise to EXPRESSION INDETERMINATE monitor printing rather than DIVISION OVERFLOW Supervisor monitor printing, for example, if expressions such as A5Q0 or A5QA2 where A2=0 are encountered.

Example:

A program being compiled attempts to print a title on an undefined output stream, causing fault printing by both Supervisor and compiler. Compiling continues, and further faults are found until the error density is too great.

```
OUTPUT NOT DEFINED
INSTR  787975  122,63 ,24 ,-1048576  B63 =-1048575  B24 = 791293
INSTR  787976 1060,0  ,61 , 16       B61 =-6
INSTR  787977  300,0  ,0  , 788005
B1  = 512        B3  =-127        B4  = 0.1        B7  = 787202
B8  = 787372.4   B9  = 257.2      B10 = 2.4        B12 =-126
B14 =-0.1        B15 = 786432     B16 = 917373.4   B17 = 917373.4
B18 = 917370.4   B20 = 911.4      B24 = 791293     B28 =-1048576
B45 = 2.1        B61 =-6          B63 =-1048575    B70 = 787583
B71 = 917371     B72 =-1          B73 = 132.4      B78 = 128.1
B79 = 917476.4   B81 = 917372.4   B88 = 917373.4   B90 = 787986
B91 = 12.2       B92 = 786740.2   B93 = 786740.2   B94 = 10
B96 =-1044754    B97 =-130560.1   B98 = 262656.5

ABL MONITORING

1,2 AO/0 ?
T10

1,9 AO/0 SHIFT >23 PLACES
H1U24

0,10 AO/0 * OUT OF RANGE
121,2,0,A5680

4,10 AO/0 PARAMETER OR ROUTINE NO. TOO BIG
121,2,0,A5680

1,11 AO/0 EXPRESSION INDETERMINATE
P1OM7

4,13 AO/0 INSTRUCTION?
121,0,0,0,0

0,14 AO/0 IRREGULAR FUNCTION
123456βP01UY

1,15 AO/0 WRONG FORMAT
F,K1,+,+4(-1,-1β,+3:0

3,15 AO/0 WRONG FORMAT
F,K1,+,+4(-1,-1β,+3:0

TOO MANY ERRORS
```

Chapter 12

FURTHER FACILITIES AND TECHNIQUES

For most purposes, the information given in the earlier chapters is sufficient to allow adequate and efficient programs to be written. Occasionally, however, it may be possible to increase the efficiency of program writing or execution; the following sections describe how this may be effected.

12.1 Programmed Drum Transfers

In the great majority of programs, the user will wish to take advantage of the one level store concept and will regard the core store and drums as a single, large main store. Programs are written as if the entire store were core store, and the Supervisor will automatically control the transfer of 512-word blocks between the drums and the core store as needed.

However, circumstances can arise in which it is useful to exercise some degree of control over these block transfers, both to ensure that blocks of information are already available in the core store when required, and to clear space in the core store by releasing blocks to the drums as soon as they are no longer needed; the extracodes provided for these purposes are all designed to assist towards greater economy of time by the avoidance of unnecessary Supervisor drum transfers. To understand just how the programmer may assist the Supervisor, it is necessary to consider the means provided for the regulation of automatic drum transfers.

In addition to any store location explicit in each instruction, there is implicit a store reference to the location containing the instruction word itself; in either case, the address is taken to specify both a block and a word within that block. The block address is invariably interpreted as a store request, and the Supervisor will initiate a drum transfer if the block is not already in the core store. Normally there will be only one copy of a particular block, occupying either a page in the core store or a sector on one of the drums. With each 512-word page of the core store there is associated a Page Address Register (P.A.R.) containing the number of the block occupying the page at any particular time; there is also a lock-out digit which is set whenever the page is involved in a drum or peripheral transfer and so is not available to the main program. At every store request, the block address is automatically compared with the content of each P.A.R.; if a coincidence is found, the store reference is completed by the extraction of the required word from the appropriate page. Otherwise a non-equivalence interrupt occurs and the Supervisor drum transfer program is entered; this is in two parts, one to carry out the actual block transfers and the other to decide which page of information should next be transferred to a drum to make space available in the core store.

All requests for information transfers between the core and the drum stores, whether originated by the Supervisor or called for directly by an object program, are placed in a drum queue holding up to 64 entries which are dealt with in the order of their occurrence. The drum transfer routine is re-entered repeatedly until the queue is cleared.

It is arranged that there shall always be at least one free page in the core store, so that, whenever the drum transfer routine is entered, the first 'read' request in the drum queue can be implemented. Then, whilst this transfer is taking place, the drum transfer learning program decides which page may next be freed by writing its contents away to a drum. This decision is made on the basis of the frequency of past references to each block of information, and with the intention of choosing the core store page least likely to be referred to.

The drum learning program only attempts to predict future store needs in the light of past requests; it anticipates neither the termination of references to a particular block of information nor the imminent requirement for a new block. Hence, there arise in the main two ways of assisting the Supervisor by means of programmed drum transfers; firstly by releasing core store pages no longer required, and secondly by initiating the reading of a new block of information from a drum to the core store before it is actually referred to. These are both of marked advantage to the system as a whole: the first plainly helps towards efficient utilisation of the available facilities, and the second can often prove of even greater benefit and economy by reducing the time spent in waiting for drum transfers to be completed - this is especially significant in the inner loops of a program. It will be appreciated that the time during which a program is held up waiting for a drum transfer is still wasteful, notwithstanding time-sharing, since it may take from 1.3 to 2.7 milliseconds, depending on the core store size, to switch to another program and a similar length of time to switch back later.

Ideally, a 'read' transfer should be timed to reach completion only just before the first reference is made to the block; otherwise the Supervisor may choose to write the block away to the drum again before the program comes to use it. The actual transfer of a block of 512 words takes 2 milliseconds, but there is an initial delay of up to 12 milliseconds, the revolution-time of the drum.

The core store of Atlas is arranged in 4096 word stacks, with 16 pages of information sharing each pair of stacks, and each stack having its own access equipment; to take advantage of this, and so to attain maximum speed, operands and instructions should be arranged, as far as is possible, in different pairs of stacks. At Manchester, the Supervisor endeavours to read down instructions to pages 0 to 15 and operands to the remaining pages of the machine; in the event of a non-equivalence interrupt the preference is automatic, being determined by the non-availability in the core store of an operand or an instruction as the case may be; in the case of a programmed drum transfer, the preference must be indicated by affixing a bit 1 before the address of a block of instructions, and 0 before the address of a block of operands. This preference bit will be the most significant bit of n (singly modified) or of ba where appropriate. At the other installations, where there is more store, instructions and operands are placed in different pairs of stacks whenever possible. Pages 0-15 form one stack pair, as do pages 15-31, 32-47, etc.

Extracodes are available for the purposes we have discussed and these will now be described with the help of the following notation:

Block address,	$P = P_1$ = bits 1 to 11 of n
Block address,	P_2 = bits 1 to 11 of ba
Number of blocks,	K = bits 21 to 23 of n
Logical band number,	D = bits 13 to 20 of n
Band or page number,	d = bits 13 to 20 of ba
Section number,	k = bits 21 to 23 of ba

In all but two of the extracodes which follow, whenever information is transferred to a new block, the old block is made free. The exceptions are 1162 and 1163, where a block is to be duplicated leaving the original copy intact.

Further, when a block is quoted as the destination of an information transfer, either directly or as the result of renaming, any existing block of the same name is lost. This will apply even if the name quoted as that of the source is unallocated, and will in fact be the only action taken in such a case. Also, in those instructions referring to two block addresses, these addresses should not be the same.

1135 b91' = c and c' = n if block number ≥ ba newly defined.
Henceforth, each time a block with a number ≥ ba is newly defined by a non-equivalence, store current control in B91 and jump to n. The block number in ba occupies bits 1 to 11 and the remaining bits of ba are ignored. The contents of Ba are undisturbed. The instruction causing the non-equivalence is not executed. n is singly modified.

1155 ba' = smallest block label ≥ n defined.
Place in bits 1 to 11 of ba the smallest block number ≥ n which is defined for this progrm. The remaining bits of ba are left cleared. Only bits 1 to 11 of n are used. n is singly modified. If all the program's blocks are < n, then bit 0 of ba' is made 1 and the remaining bits are cleared.

1160 Read block P.
If P is not already in the core store, the transfer request is inserted in the drum queue exactly as if a non-equivalence interrupt had occurred, but control is restored to the object program immediately the drum queue entry has been made. Should the queue be already full, the object program will be halted until the entry can be inserted.

1161 Release block P from the core store.
This extracode adjusts the parameters used by the drum learning program so as to cause it to choose block P next for writing away to the drum store, if this has not already occurred. No entry is made in the drum queue and the transfer will in general take place earlier than if extracode 1165 (below) had been used.

1162 Duplicate block P_1 as P_2 in the core store.
Any existing block P_2 is always lost, and, if P_1 is allocated,

a copy of it will be formed as P_2 in the core store. Unless the drum store is full, block P_1 will finally be located there; otherwise P_1 will be left in the core store. $P_1 \neq P_2$.

1163 Duplicate block P_1 as P_2 in the drum store.
Provided P_1 is allocated, the effect of this extracode is to form a duplicate copy of it. It will be arranged that one copy shall always be left in the core store and named P_1; the second copy, named P_2, will be put in the drum store unless this is full, in which case it will be left in the core store. Any previously existing block P_2 will be lost in all cases. $P_1 \neq P_2$.

1164 Rename block P_1 as P_2.
If P_1 is allocated, the appropriate entry in the drum directory or the core store P.A.R. is altered to P_2. Any P_2 previously existing will be lost. There must be at least one more block allowed for in the job description than is defined at that moment. $P_1 \neq P_2$.

1165 Write block P.
Provided P is allocated and is not already on a drum, it is transferred to the next empty sector. Should the drum store be full, block P is released, precisely as in 1161.

1166 Read block P to absolute page d.
This extracode makes possible full control of the store by those exceptional programs for which this may be worthwhile. Before using 1166, the program must set a trap in case page d is locked down and reserved by the Supervisor.
d is in the integer position of ba and defines the absolute number of a page in the core store to which block P is to be transferred. Before this transfer takes place, any existing contents of d are copied to a free page.

1167 Lose block P.
If P is allocated, the page or sector occupied by it is made free.

1170 Clear new blocks/Do not clear new blocks.
When a program refers to a main store block for the first time, the Supervisor allocates a free page of the core store; floating-point zero will be written in all 512 words if the clear blocks switch is set. Initially, this switch is set to clear all new blocks, but it may subsequently be set or reset by means of extracode 1170 according to the sign bit of n:-

$n \geq 0$	Clear new blocks.
$n < 0$	Do not clear new blocks.

1171 Change store allocation to n blocks.
Each program has some number of main store blocks assigned to it. This number may be altered during the execution of the program by the use of extracode 1171. If there are less

than n blocks available in the store, then the program will be faulted for ILLEGAL FUNCTION and EXCESS BLOCKS.

1172 Set ba' = number of pages available.
This extracode provides an estimate of the number of core store pages available to the program at a particular moment. It cannot be assumed that this number of pages will continue to be available, since the core store allocations are always fluctuating.

1173 Set ba' = number of blocks available.
At a particular moment, this extracode records the maximum number of main store blocks available, consisting of all unallocated blocks together with those already allocated to the program itself.

1174 Reserve band D.
A complete band of the drum store is reserved for the program and may subsequently be referred to as band D.

1175 Read K + 1 blocks from band d, starting at sector k.
d must already be defined by extracode 1174. The K + 1 successive sectors k, k + 1,..., k + K are read to store blocks P, P + 1,..., P + K. If K is 6 (or 7), sectors k (or k and k + 1) will be read twice. Thus if K = 6, blocks P and P + 6 will both contain sector k. If k is 6 or 7, it is taken as 0 or 1 respectively. All blocks involved are locked down until the entire transfer is complete.

1176 Write K + 1 blocks to drum band d starting at sector k.
d must already be defined by extracode 1174. This extracode writes store blocks P, P + 1..., P + K to drum sectors k, k + 1,..., k + K. Sectors 6 and 7 are the same as sectors 0 and 1. If K exceeds 5 some of the earlier blocks are overwritten. Thus if K = 6, sector k will finally contain block P + K rather than block P.

1177 Lose band D.
The band of the drum store previously reserved as logical band D is freed and made available for general use.