903/920 SIR

Book No. 103

Copy No. 23

Amendment No. 3

AIRBORNE COMPUTING DIVISION
ELLIOTT FLIGHT AUTOMATION LIMITED

PREFACE.

This book describes the following tapes:-

1-PASS SIR,	24/3/71,		Binary Mode 3
2-PASS SIR,	7/1/71,		Binary Mode 3
1-PASS SIR	Options	(Mylar)	
2-PASS SIR	Options	(Mylar)	

They enable machine-code programs written in symbolic form to run on any 903, 905 or 920 computer having tape reader Mode 3.

The above 1-PASS SIR and 2-PASS SIR both operate in 900-Series Telecode and 920 Telecode.
This book does not include
tapes which operate in one code only.

The tape reader modes and Telecodes refered to above and within this book, and the A.C.D. Binary tape format and A.C.D. Internal code referred to in this book are defined in Book 106 '903/905/920 USEFUL NOTES'.

The reader who is unfamiliar with 903, 905, or 920 machine-code and SIR programming is recommended to read Book 113, '903 SIR PROGRAMMER'S GUIDE' and Book 101 '903/920 SIR Course Notes'.

1 - PASS SIR, 24/3/71, Binary Mode 3

2 - PASS SIR, 7/1/71, Binary Mode 3

Contents

PREFACE.

These notes describe the SIR assemblers, and assume that the reader is familiar with 903/905/920 machine-code.

These assemblers can be operated on any 903, 905, or 920 computer with an 8192-word store and a high-speed punch and reader. The 2-pass assembler also contains facilities for writing programs for a 16384-word store.

Both assemblers can be operated in 900-Series Telecode or 920 Telecode.

Both assemblers normally use reader mode 3 throughout, (although mode 1 is required to reload intermediate (R.L.B.) tapes made by earlier issues of 1-PASS SIR)

The 1-pass assembler is used during the development of a program of up to about 4000 or 5000 words. For larger programs, or to assemble a completed program of any length, the 2-PASS assembler is used.

CAUTION.

The issue of 2-PASS SIR described in this book has been used to MAKE tapes for loading into a 16384-word store, but no opportunity has arisen to CHECK these tapes.

Chapter 1: INTRODUCTION

1.1 General

The Symbolic Input Routine (SIR) enables programs to be written in a modified form of machine code which has two principal advantages over machine code:

(i) It is not necessary to specify the absolute address of a store location used in a program. Locations may instead be referred to by names invented by the programmer and the SIR assembler will allocate a specific store location for each such invented name.

(ii) It is possible to write instructions using constants, without specifying where the constant is stored. Instead the constant itself is written in the address part of the instruction.

Programs written in SIR code can be assembled by means of the 1-PASS SIR assembler in two ways, load-and-go and non load-and-go.

(iii) Programs assembled in load-and-go mode are loaded into the computer ready for triggering.

(iv) Programs assembled in non load-and-go mode, however, are output in a relocatable binary code so that they can be entered into the computer by means of the SIR binary loader routine within 1-PASS SIR. The reason for having this alternative mode of assembly is given in chapter 11.3.

Programs written in SIR code can also be assembled by 2-PASS SIR.

1.2 Glossary of terms.

In the following glossary a brief explanation of each term is given followed where necessary by a reference to a chapter where a full definition or explanation is to be found.

ALPHANUMERIC CHARACTER any tape character which has a six bit internal code representation (6.3)

ALPHANUMERIC GROUP a group of three ALPHANUMERIC CHARACTERS - a type of constant (6.3)

ASSEMBLER the program which reads and translates programs written in SIR code (12.1)

BLOCK the main division of a PROGRAM: It comprises a GLOBAL IDENTIFIER LIST followed by a CODE BODY (Chap. 3); and should be preceded by a TITLE

BLOCK RELATIVE ADDRESS (N;) the address of location N of the current. BLOCK, where N is an unsigned integer. (The first location of a BLOCK is relative location zero) (5. 2. ii). (obsolete)

CODE BODY all that part of a BLOCK other than the GLOBAL IDENTIFIER LIST. It includes constants, instructions and work-space (Chap. 3).

COMMENT information inserted in a SIR program which may be meaningful to human beings, but is ignored by the ASSEMBLER. Comments are enclosed in round brackets ( ) (Chap. 8). See Also TITLE

CURRENT PLACING ADDRESS (CPA) the address where the next word will be placed by SIR (10. 1). also called STORE POINTER (SP).

CURRENT PLACING ADDRESS RESERVE (CPAR) a location holding a former placing address used in conjunction with the PATCH and RESTORE facilities (10. 1).

DECLARATION the use of an IDENTIFIER as a LABEL (Chap. 4).

DICTIONARY the part of the computer store in which the ASSEMBLER keeps a list of IDENTIFIERS, INCREMENTS and LITERALS together with references to the locations to which they refer. Also the list itself.

DIRECTIVE a PATCH, RESTORE, SKIP or OPTION (qqv.). Directives tell the ASSEMBLER how and where it is to store the translated program.

GLOBAL IDENTIFIER an IDENTIFIER having the same meaning in several PROGRAMS (4. 2).

GLOBAL IDENTIFIER LIST the list of GLOBAL and SUB GLOBAL IDENTIFIERS, valid in the BLOCK it heads, that is enclosed in square brackets and occurs at the head of each BLOCK (4. 12).

HALT CODE a character punched on a SIR mnemonic tape, at the beginning of a newline, which causes the ASSEMBLER to wait. Can be written H

IDENTIFIED ADDRESS an address consisting of an IDENTIFIER alone or an IDENTIFIER followed by an INCREMENT (5. 3).

IDENTIFIER an invented name used as substitute for an address (4).

INCREMENT a signed integer following an IDENTIFIER to modify its meaning (5.3).

INTERMEDIATE TAPE : see RELOCATABLE BINARY TAPE

LABEL an IDENTIFIER preceding a word and referring to the location containing that word (4. 3).

LABEL LIST a list of LABELS together with their addresses which can be punched during ASSEMBLY (11. 2).

LITERAL a constant appearing as the address part of an instruction (5. 4).

LOAD-AND-GO a mode of operation in which a SIR program is assembled into the computer store for immediate use. cf. NON LOAD-AND-GO (11. 1, 12. 1).

LOADER a tape read in by the initial instructions, as punched at the start of 2-PASS SIR binary tapes.

LOCAL IDENTIFIER an IDENTIFIER which retains its meaning only inside the block in which it is declared (4. 1).

NON LOAD AND GO a mode of operation in which a SIR program is translated to a RELOCATABLE BINARY TAPE (11. 2, 12. 1).

OPTION (*N) a DIRECTIVE to the ASSEMBLER which enables the programmer to vary the way in which the assembler operates (Chap. 11).

PATCH (↑N) a DIRECTIVE used to correct or control the placing of a SIR program. It instructs the assembler to store program in location N onwards (10. 1).

PERCENT SIGN(%) the end of program symbol. On reading it the ASSEMBLER locates constants and checks for undeclared identifiers and then waits (9. 2).

PROGRAM a sequence of BLOCKS terminated by a PERCENT SIGN.

PSEUDO INSTRUCTION an instruction not intended to be obeyed. It is used as, for example, a constant. It is written in an identical form to other instructions (6. 4).

QUASI-INSTRUCTION a literal address in the form of an instruction (5. 4. 1).

RELOCATABLE BINARY (RLB) TAPE a special tape holding a SIR program which is output in NON LOAD AND GO assembly (12. 1, 12. 2).
Also called an INTERMEDIATE tape.

RESTORE ($) a DIRECTIVE which cancels the effect of a PATCH or series of PATCHES by restoring the placing address to its original value (10. 1).

SEPARATOR a space tab or new line. It is used to separate different SIR elements. Can be written S, T, or N.

SIX-BIT INTERNAL CODE the code in which the ASSEMBLER stores characters three to a location. See code table in (1. 5).

SKIP (> N) a DIRECTIVE, normally used to reserve work space, which instructs the assembler to leave the next N store locations unaltered (7).

STORE POINTER: See CURRENT PLACING ADDRESS

SUB GLOBAL IDENTIFIER an IDENTIFIER having the same meaning in several BLOCKS (4. 1).

TITLE a COMMENT at the start of a tape or BLOCK enclosed between a double bracket ((and a single bracket) for the purpose of identifying a programme or block. (Chap. 8)

1. 3 Elements of SIR Programs

The following basic elements may occur in a SIR program, and must be separated from each other by at least one separator.

Words	Labels
Patches, Restores,	Global Identifier Lists
Skips	Stop Codes
Options	Percent symbols
Comments	A Trigger.

See references in 1. 2 for details of these elements.

1. 4 'New line' sequence

The SIR assemblers read one line of source text at a time into a read buffer. Every new line should be followed by several blanks to simplify future editing of the tape. (The omission of these blanks is not an error). To this end edit programs are available to automatically insert blanks after each new line.

Note that these blanks are essential if stop-on characters are not used.

On 900 code tapes, newline may consist of 2 characters, carriage return & line feed. SIR ignores the former and treats 'line feed' as 'newline'.

1. 5 Six bit internal code.

6-bit code Decimal	Octal	Character	6-bit code Decimal	Octal	Character
0	00	Ⓢ Space	32	40	` grave
1	01	Ⓝ Newline	33	41	A
2	02	"	34	42	B
3	03	£	35	43	C
4	04	$	36	44	D
5	05	%	37	45	E
6	06	&	38	46	F
7	07	´ acute	39	47	G
8	10	(	40	50	H
9	11	)	41	51	I
10	12	*	42	52	J
11	13	+	43	53	K
12	14	,	44	54	L
13	15	-	45	55	M
14	16	.	46	56	N
15	17	/	47	57	O
16	20	0	48	60	P
17	21	1	49	61	Q
18	22	2	50	62	R
19	23	3	51	63	S
20	24	4	52	64	T
21	25	5	53	65	U
22	26	6	54	66	V
23	27	7	55	67	W
24	30	8	56	70	X
25	31	9	57	71	Y
26	32	:	58	72	Z
27	33	;	59	73	[
28	34	<	60	74	£
29	35	=	61	75	]
30	36	>	62	76	↑
31	37	$_{10}$	63	77	←

Notes about the 6-bit code.

1. Ⓝ may be punched as "new line" or "carriage return + line feed". See paragraph 1.4.

2. On input no distinction is made between upper case and lower case letters. Letters are always output in upper case.

3. On 920 Telecode tapes, $\not{S}$, ´acute, and `grave may be punched as \$, ¢, and $\not{b}$, using the non-escaping vertical bar character, and the symbol " may be punched as ~.

4. On 900-Series code tapes, `grave may be punched as either `grave or @, £ may be punched as £, ½, or \, and ₁₀ suffix may be punched as ?.

5. The symbol Ⓣ, horizontal tab, may also be used in SIR tapes, and is treated as Ⓢ.

6. This 6-bit SIR internal code is simply related to "A.C.D. Internal Code, 1/12/69" described elsewhere.

Chapter 2: WORDS

Words are the basic elements of a SIR program. After assembly each SIR word occupies one store location in the computer. Words can be written in two forms:-

(i)	constants	e.g.	+304 -.2667
(ii)	instructions	e.g.	15 2048 /2 CAT+10

All words must be followed by a separator. Words are entered into consecutive store locations in the order that they appear in the SIR program. The only time that the assembler does not obey this rule is when it receives an order to the contrary in a directive (patch, skip, or option).

Chapter 3: BLOCKS

Every SIR program consists of one or more blocks. Each block is divided into two parts:-

(i) A Global Identifier List which is enclosed by square brackets [ ]. This part of the block may only contain identifiers and separators.

(ii) A Code Body which follows the Global Identifier List of the same block, and which is terminated either by the [symbol at the start of the next block, or by the end of program symbol (%).

The significance of these terms is explained in the next chapter.

The last instruction capable of being obeyed in each block must be an unconditional jump (e.g. the dynamic stop 8;+0 explained in 5.2(i) below). It will usually be followed by labelled constants and work space.

The effect of trickling out of the end of a block is undefined.

Example

```
[PRINT INT FRAC]                Global Identifier List  ┐
PRINT  >1           ┐                                   │
INT     5 FRAC      │                                   │
        . . . . .   │  Code                             ├ block 1
        . . . . .   │  body                             │
        0 PRINT     │                                   │
       /8 2         │                                   │
WSP    >1           ┘                                   ┘
[INT]                                                   ┐
. . . . . . .                                           ├ block 2
. . . . . . .                                           ↓
```

N.B. The Global Identifier List may be omitted at the head of a one-block program, but if it is omitted block relative addresses (see 5.2(ii)) may not be used.

The use of titles before each block (Chapter 8) is strongly recommended.

Chapter 4: IDENTIFIERS

An identifier is a name invented by the programmer which is a substitute for an address. Any combination of letters A-Z and digits 0-9 is acceptable as an identifier, provided that the identifier starts with a letter.

e.g.

A HOUR T52 MULTIPLE	are acceptable identifiers

32BIT	are	starts with a digit
BL(BATH)	not	parentheses
T 52	acceptable	space
B-LINE	because	hyphen

Identifiers are distinguished from each other by their first six characters. Thus no distinction is made between FLIGHT, FLIGHT1, FLIGHT2 and FLIGHTPATH.

Since no distinction is made between upper and lower case letters the identifiers FLIGHT, flight, fLiGhT and Flight are treated as identical. Programmers are strongly advised to use upper case exclusively when writing programs, except for comments.

Identifiers are declared by being used as labels. Consequently every identifier must be used as a label once and only once within its range of validity.

4.1 Global and Sub-Global Identifiers.

4.1.1 Global Identifiers.

Global Identifiers are the links between the different blocks of a program. They must be listed in the Global Identifier Lists at the head of the block in which they are declared and at the heads of every other block in which they are to be valid. One or more separators must follow each identifier in a Global Identifier List, and only identifiers separators and the Sub-Global Identifier marker, " or ~, may occur between the square brackets which enclose the list.

When an identifier is included in the Global Identifier Lists of two or more blocks which are assembled together it refers to a single address indicated by a label in one of these blocks (the block in which it is declared). An identifier which is used globally in some blocks may be used as a local identifier in any block in which it is not listed as global.

4.1.2 Sub-Global Identifiers.

If on its first occurrence in a Global Identifier List an identifier is immediately preceded by " or ~ it is treated as a Sub-Global Identifier.

Whereas a Global Identifier remains in the SIR* dictionary after the end of program symbol % has been encountered and permits communication between several programs that are in store together, Sub-Global Identifiers are removed from the SIR dictionary when % is encountered.

The listing of an identifier as Global or Sub-Global is determined by the first Global Identifier list in which it occurs and is valid for a complete program. An identifier cannot be Global in some blocks of a program and Sub-Global in others.

4.1.3 Examples.

[MOUSE "HAMSTER "LION WOOLF]

MOUSE and WOOLF are Global Identifiers.

HAMSTER and LION are Sub-Global Identifiers.

4.2 Local Identifiers.

Identifiers which are neither Global nor Sub-Global are Local. Local identifiers have no meaning outside the block in which they are declared.

The same name may represent a Global or Sub-Global Identifier in some blocks and several different Local Identifiers in other blocks and be undefined elsewhere (see 4.5).

4.3 Labels and Declarations.

Each Local Identifier is declared by being used once and only once as a label in the block for which it is valid.

Similarly each Global or Sub-Global Identifier is declared by being used once and only once as a label in exactly one of the blocks for which it is valid.

Labels are followed by one or more separators, and refer to the store location into which the word following the label is to be assembled.

```
e.g.    OUTPUT  15 6144
        AREA    -23378
```

---

*1-PASS SIR Only. 2-PASS SIR ~~treats~~ subglobal identifiers as global.

A location may be labelled by several identifiers with one or more separators between them. They need not be all on one line.

```
e.g.    8 REPEAT
        BEGIN GO START
        ENTRY
                  4 FLAG
```

Assume that the instruction 8 REPEAT is assembled in location 2300. Then BEGIN, GO, START, ENTRY all refer to location 2301. If, however, 8 REPEAT was assembled in 2336, then BEGIN, GO, START, ENTRY would refer to location 2337, into which the instruction 4 FLAG would be assembled.

An absolute address may be labelled. Such a label is written thus: CONTINUE=9. This would allow location 9 to be referred to as CONTINUE.

4.4 Example (see Figure 2).

(i) Programs are named after the Global or Sub-Global identifier that labels their first instruction.

(ii) Blocks are named after the Global or Sub-Global identifier that labels their first instruction.

(iii) APPLE is Global in both programs.

(iv) PEAR is Sub-Global in program APPLE and another PEAR is Sub-Global in program PLUM.

(v) APRICOT is Global in both programs and another APRICOT is Local to block PEAR of APPLE.

(vi) ORANGE is Local to block APPLE of APPLE.

(vii) PLUM is Sub-Global in program APPLE and another PLUM is the name of the second program.

(viii) PRUNE and END are Local in block PEAR of PLUM.

(ix) A third program could refer to the Global Identifiers APPLE, APRICOT and PLUM.

(x) The example is a nonsense program.

```
    [APPLE "PEAR APRICOT]
APPLE    0       APRICOT
        /4       17
ORANGE ≡ 192                         block         program
         7  PEAR                     APPLE
         8  ORANGE                                 APPLE
    [PEAR  APPLE  "PLUM]
PEAR    10  APRICOT
         4  APRICOT                  block
         9  PLUM                     PEAR
         8  APPLE
APRICOT +0
    [PLUM  APRICOT  PEAR]
     PLUM 8 PEAR                     block
     APRICOT >100                    PLUM
%


    [PLUM APRICOT "PEAR]
PLUM ≡ 5095
  ↑PLUM                              block
  0 APRICOT                          PLUM
 10 17                                             program
  8 PEAR                                           PLUM
    [PEAR  APPLE]
PEAR 10 PRUNE
      4 PRUNE                        block
      7 APPLE                        PEAR
END   8 END
PRUNE -5
%
```

Chapter 5: INSTRUCTIONS

Words written in the form of instructions are introduced by a / (B-line) or a digit. Each word comprises two parts, a function and an address, separated by one or more separators.

The functions consists of a decimal integer between 0 and 15 representing corresponding functions and a / symbol immediately preceding the integer if B-modification is desired.

The Address part of an instruction can be written in four different ways; Absolute, Relative, Identified or Literal. It is assembled as an integer in the range 0-8191 and is interpreted at run-time as relative to the start of the store module in which the instruction is placed.

References to locations in other store modules are made by means of B-lined instructions.

5.1 Absolute Addresses.

An absolute address is an unsigned integer not greater than 8191, and it refers to the computer store location with that integer as its address. In functions 14 and 15, however, the absolute address gives further specification of the function by the usual conventions.)

Examples

4 8180	load the accumulator with the contents of location 8180
15 6144	punch (the least significant 8 bits of) the contents of the accumulator

5.2 Relative Addresses.

(The relative addresses must not be outside the range 0-8191). Relative Addresses are of two kinds:-

(i) Location Relative Address consisting of a semicolon followed by a signed integer. This address refers to a location, the address of which, is the sum of the address of the location in which the current instruction is being assembled, plus the signed integer.

e.g. 7 ;+3 means "jump three locations forward if zero"
5 ;-1 means "store in the previous location"
8 ;+0 means a dynamic stop.

Note that 8 ;0 is not a permissible instruction.

(ii) Block Relative Addresses consisting of an unsigned integer not greater than 8191 followed by a semicolon. This address refers to a location with address equal to the sum of the unsigned integer and the address of the first location of the current block.

e.g.
```
[MASS]
 +336
 4 MASS
 5 30;
```

If +336 is assembled in location 3000 then 5 30; is assembled as 5 3030. This facility is provided for compatability with an earlier assembler only.

5.3 Identified Addresses.

An identified Address consists of an identifier alone or followed by a signed integer. An identified address is introduced by a letter. The assembler replaces the identified address by the sum of the address of the unique location labelled by the identifier, plus the signed integer. The signed integer is called an increment even if it is negative. Thus, in Example 1, the instructions 5 CAT+10 and 4 CAT-3 are both incremented instructions with increments +10 and -3 respectively.

An identified address may be used in the text before the identifier to which it refers has been declared, i.e. has appeared as a label.

Although an incremented identifier may be referred to before it has been declared, such references greatly increase the amount of workspace required by the SIR* assembler itself. Consequently, if there is a block of global work space it should be declared early in the program and, arrays of local workspace should be declared near the start of the block in which they occur. This has been done in Example 2.

Example 1

```
CAT 4 WS2
    4 FLAG
    7 ERROR
    4 CAT
    5 CAT+10
    4 CAT-3
```

If 4 WS2 is assembled in location 5600 then
4 CAT is assembled as 4 5600
5 CAT+10 is assembled as 5 5610
4 CAT-3 is assembled as 4 5597

*1-PASS only.

Example 2

```
                 [MXMULT]
                 8 MXMULT
         MATRIX  > 400          (COMMENT THIS IS A SKIP)
                 +0
         MXMULT  4 WS1
                 . . . . . . .
                 4 MATRIX+265
                 . . . . . . .
```

If 8 MXMULT is assembled in location 3072 then 4 MATRIX+265 is assembled as 4 3338 (3338 = 3072+1+265). The use of skips is explained in Chapter 7.

5.4 Literal Addresses.

Literal addresses are introduced by +, -, =, &, or £. They are used to make it easier to write instructions which operate on constants. Instead of putting in the address part of the instruction an identifier which labels the constant at some other point in the program

```
e.g.        TEN     +10
                    .......
                    4 TEN
```

the programmer may put the constant itself into the address part of the instruction;

```
e.g.        4       +10
```

The assembler makes a special note of this. On reading the end of program symbol % (see 9.2) it allocates a store location in which it places the constant and inserts the address of this location in all the instructions which use it.

There are four types of literals, corresponding to the four different possible types of constants.

(i) integers and fractions
(ii) octal groups
(iii) alphanumeric groups

these have exactly the same form as the corresponding constants.

```
e.g.  4 -.2667      6 &7777
      2 +360        4 £E26
```

(iv) quasi-instructions (see below)

5.4.1 Quasi-instructions.

Quasi-instruction literals differ in two respects from pseudo-instruction constants.

(i) every quasi-instruction is introduced by an = sign which immediately precedes the function bits or the solidus indicating B-modification if this is present.

(ii) the address part of a quasi-instruction must be in absolute form - relative, identified or literal addresses are given as errors (error E0)

Examples

4	=8 0	
4	= /0 0	i. e. zero accumulator except for 1 in sign bit.
6	=15 8191	i. e. make sign bit zero.*

one space exactly is recommended here.

Note: Literal addresses may only be used with functions 0, 1, 2, 4, 6, 12, 13. If an attempt is made to use a literal address with any other function the error message EL will be displayed.

---

* This is less pathologically written 6 &377777. (See Chap. 6.2)

Chapter 6: CONSTANTS

There are four types of constants allowed in SIR:

(1) Integers and Fractions
(2) Octal Groups
(3) Alphanumeric Groups
(4) Pseudo-Instructions.

All constants must be followed by a separator.

6.1 Integers and Fractions.

An integer or fraction is introduced by a + or - sign. If the + or - sign is immediately followed by an integer, then the constant is stored as a binary integer.

e.g. +14 stored as 000 000 000 000 001 110
-64 stored as 111 111 111 111 000 000

Integers must be in the range -131,071 to +131,071 inclusive. (-131,072 may be written as the octal group &400000)

If the + or - sign is immediately followed by a decimal point (.) followed by an integer, the constant is stored as a binary fraction.

e.g. +.375 stored as 001 100 000 000 000 000 *
-.5 stored as 110 000 000 000 000 000

(The 'fraction' -1 can be written in the same way as the integer -131072)
Fractions may contain up to six digits.

6.2 Octal Groups.

Octal Groups are introduced by a '&' sign. An 18-bit word can be divided into 6 groups of 3 bits, each being equivalent to a digit from 0 to 7. Thus a constant can be written as a group of 6 octal digits, which immediately follow the '&' sign.

e.g. &312705 is equivalent to 011 001 010 111 000 101

Octal groups of less than 6 digits can occur, in which case they are right-hand justified (i.e. &42 means the same as &000042).

* 2-PASS SIR also allows fractions to be followed by a power of 2 scaling factor as in 0.12/+2 (= 0.48) -12.63/-4 (= 0.5777...)

6.3 Alphanumeric Groups.

Alphanumeric groups are introduced by a £ sign, which is followed by up to three alphanumeric characters. These are packed, from left to right, into the store location in the 6-bit SIR internal code. All characters included in the code table can be stored except that

(i) % cannot be included, (see 10.2)

(ii) the alphanumeric group is considered as complete if a new line is encountered before three characters have been read after the £ sign. In this case the group is left-hand justified (i.e. the remaining characters are considered to have code 0, the code for a space). The new line is NOT considered as one of the characters of the group, but instead acts as any ordinary separator.

(iii) Spaces, however, when occurring in the three characters following a £ sign, are treated like any other character.

(iv) "Tab" in alphanumeric groups is treated as "space".

(v) To enable the internal code of "newline" to be stored, (despite (ii) above) the symbol ↑ in alphanumeric groups is stored as value &01.

(vi) £ is stored as value &03, whether punched as £, ½, or \.

The chief use of alphanumeric groups is for storing characters which are to be punched out at some stage of the program. This can only be done if the program also contains a print routine and a table for conversion from internal to external code.

Any of the character subroutines used to output "A.C.D. Internal code 1/12/69" will be found useful for printing alphanumeric groups.

Examples

Alpha-numberic group as written	Actual octal equivalent	Form placed in store	
		in octal	alphanumeric equivalent
£MAN	55 41 56	55 41 56	MAN
£ space=new line	00 35 xx	00 35 00	space = space

(xx indicates an unspecified character)

6.4 Pseudo-Instructions.

These take the form of instructions but are used as constants. They are identical in form to ordinary instructions.

e.g. /0 0 can be used to represent the integer - 131,072

Similarly, constants can be obeyed as instructions. The intentional use of constants in this manner is frequently described as pathological programming and is to be deprecated. Failure to terminate an instruction sequence with an unconditional jump as described in Chapter 3 is liable to result in this unwanted effect.

Chapter 7: SKIPS

A skip >, indicates that a number of store locations are to be left unaltered before the assembler continues filling the store with SIR words. The number of locations which are to be left unchanged is indicated by an optional + sign and an integer which immediately follows the > sign.

For example, if the following piece of SIR program occurred.

```
+133
> 15
4 8180
5 COUNT
........
```

and the word +133 was entered into location 5000 in the computer store, the skip > 15 indicates that the next word, the instruction 4 8180, is to be assembled not in location 5001 but in location 5016, the instruction 5 COUNT is then assembled in location 5017 and so on.

The chief use of skips is to reserve locations for work space without assigning any values to them.

7.1 Labelled Skips.

Locations left unchanged by skips may be labelled in the same way as locations occupied by words.

e.g.

```
         8 ERROR
         > 4
ALPHA    > 10
MATRIX   > 400
BETA     > 10
```

In this case if 8 ERROR is assembled in location 4000, ALPHA refers to location 4005, MATRIX to location 4015, BETA to location 4415.

Notes:

1. The last word of the 10-word vector labelled ALPHA is addressed as ALPHA+9. Similarly for MATRIX and BETA.
2. Addresses outside the range indicated in note 1 may, of course be referred to by incremented instructions. Thus ALPHA+11, MATRIX+1 and BETA-399 are alternative ways of referring to the second location of the array MATRIX. However the increment relative to ALPHA would have to be changed if the length of ALPHA was changed and the increment relative to BETA would have to be changed if the length of MATRIX was changed.

Chapter 9: END OF TAPE AND END OF PROGRAM SYMBOLS

9.1 End of tape symbol (halt code)

A halt code punched on a tape causes the assembler to wait. Assembly can then be continued by re-entering at CONTINUE (see chap. 12) when the next tape is under the reader.

Halt codes are chiefly used:

(i) at the end of each tape of a program punched in parts.

(ii) at the end of patches.

Frequently, when a program is being developed, each block on a tape is terminated by a halt code and several inches of blank tape.

9.2 End of program symbol (%)

On reading a % symbol at the beginning of a line the assembler displays a list of undeclared local and sub-global identifiers, locates all the literals in consecutive locations immediately following the program in the order in which they occurred in the program, displays a list of undeclared global identifiers followed by a 'FIRST LAST' message, indicating the store used by the program and waits. Further symbols on the same line will be ignored but the line must be terminated with a new-line symbol in the usual way. A % symbol should be put:

(i) at the end of the last tape of a program in load-and-go

(ii) at the end of each section of a program which is to be assembled as a separate relocatable binary tape in non-load-and-go.

It will frequently be found convenient to end all tapes with a halt code and to read the % symbol from the on-line teleprinter or from a special tape comprising the character sequence: Ⓝ % Ⓝ Ⓗ.

Chapter 8: COMMENTS & TITLES

Comments are included in a program for the sole purpose of making the print-up of the program easier to understand.

A string of characters between (and) is a comment, and is ignored by SIR. A comment may be inserted anywhere in a SIR program except in a Global Identifier List. Comments must not, however, split any SIR element.

e.g. the section of program:

```
9        ERROR2        (NUMBER OVERFLOW ERROR-
                          INTEGER>131, 071)
4        INT
5        WS2
...............
```

is assembled as if it were:

```
9 ERROR2
4 INT
5 WS2
..........
```

If the first symbol inside a comment is another "(" the comment is called a title, and will be copied by SIR onto the label list, thus providing a record of tapes loaded.

e.g. ((SQUARE ROOT SUBROUTINE)

is a title.

Titles and comments may only contain characters in the 6-bit code (see 1.5), and, of these, "%," "(," and ")" should not be used, except that (may be used for the purpose described in the previous paragraph. Note in particular that a "round bracket count" is not kept.

A title should be both preceded and followed by a N and should contain no N's or lower-case letters.

Chapter 10: SPECIAL FACILITIES

10.1 Patch and Restore.

A patch is a directive to the SIR assembler to stop placing instructions in consecutive store locations and to place them consecutively from the location indicated by the patch. At the end of a sequence of patches compilation of the main program can be continued by the directive restore.

It is the responsibility of the user of these facilities to ensure that no location, whose contents will be changed by the later action of the SIR Assembler, is altered. (Such locations normally contain in their address parts information used by SIR, changing this information may lead to the corruption of other parts of the program). Any location containing an instruction which refers to a currently unplaced literal or identifier falls into this class.

10.1.1 Patch.

A PATCH is written

↑A

where A is a constant or any currently located address. Its effect can be formally defined as

if CPAR = -1 then CPAR:=CPA

then or otherwise CPA:=A

where

CPA is the Current Placing Address, i.e. the address in which SIR will place the next item, and

CPAR is a location used to hold a copy of the CPA when inside a Patch. (CPAR is initially set to -1 by the assembler).

In non-load-and-go mode a patch may be made to an unlocated label if it is the first thing in the program, (apart from global lists and comments). No other patches, or 'Restores' ($) are allowed in the program. The label must be located when loading the RLB tape.

10.1.2 Restore

The symbol $ written by itself on a new line causes assembly to continue from the location which would have been used but for the intervention of a Patch or Patches. Its effect can be formally defined as

if CPAR ≠ -1 then CPA:=CPAR

then or otherwise CPAR := -1

The reader should work out for himself why a Patch read in after the end of a program, which uses literals must end

$
%

10.2 The Trigger facility in 1-PASS SIR.

2-PASS SIR contains a trigger facility, described in section 17.2. To enable programs containing triggers to be tested with 1-PASS SIR prior to assembly by 2-PASS SIR, 1-PASS SIR now accepts the trigger symbol "<".

The address at which the symbol "<" occurs in the program is recorded by 1-PASS SIR. When assembly has been completed, by loading a % sign, the stored program may be obeyed, starting at the recorded address, by entering 1-PASS SIR at "EXECUTE", viz. &17743.

The facility is only available in load-and-go mode; in R.L.B mode "<" is ignored.

Chapter 11: OPTIONS FOR 1-PASS SIR.

Options are used to alter the way in which the assembler operates. They are introduced by an asterisk (*) followed by an optional + sign and an integer. The last seven bits of the integer are examined and variations are made in the operation of the assembler as follows:-

Bit	Meaning if bit has the value		Availability		
	1	0	Load & Go	Non-Load & Go	Check
1	display labels	don't display labels	0 or 1	0	1
2	load and go	non load and go	1	0	0
4	clear the store	take no action	0 or 1	0	0
16	assemble from 8	continue at NEXT	0 or 1	0	0
32	set dictionary below program	set dictionary below assembler	0 or 1	0	0
64	perform checks only	compile program	0	0	1

When assembly is started at START, an option of * 3 is automatically assumed. This option, like all other options is cancelled when the next option is read by the assembler. It should be noted that the 1, 2 and 64 bits enforce conditions that hold continuously, whereas the 4, 16 and 32 bits direct the assembler to do one operation at the time that the option is encountered.

It is not possible to enforce all combinations of the options indicated by the six bits. The 2 bit is examined first to decide whether the assembly is operating in the load-and-go or non load-and-go modes, and the other bits are then examined where appropriate.

The difference between load-and-go and non load-and-go programs, and the action of the binary loader, are explained more fully in chapter 12.

11.1 Load-and-Go mode

When the 2 bit in an option has value one, the assembler operates in the load-and-go mode, i.e., it assembles the source program in the computer store ready for triggering.

All the other options are available, and the bits are examined in the following order:

(i) 16 bit (continue at 21)

If the 16 bit = 1, the assembly will continue at location 8.

(ii) 4 bit (Clear the Store)

If the 4 bit = 1, the assembler clears all locations from the one where the next word is to be assembled to just before the SIR assembler itself.

(iii) 32 bit (set dictionary below program)

The Dictionary is the area of store where the assembler lists all the identifiers and literals it finds. It is normally built up just below the SIR assembler itself, but if the 32 bit = 1, it is built downwards from the location preceding the one where the assembler is about to put the next word.

This option may be used when storing a program in the high end of store. It may not be used in the same option integer as bit 4. When option bit 32 is set = 1 the test which guards against program overwriting dictionary or the Assembler itself is removed.

(iv) 1 bit (Display Labels)

If the 1 bit = 1, whenever the Assembler finds a label it punches it on a newline, together with the octal and decimal addresses of the location to which the label refers. Local labels are preceded by 2 spaces, Subglobal labels by 1, and Globals by none.

Titles are punched on the labels list, in load-and-go mode, irrespective of the 1-bit. An extra newline is punched when a new block is found.

Note that, if error indications occur, they will appear among the labels.

11.2 Non Load-and-Go mode.

When the 2 bit has value zero, programs are assembled in the non load-and-go mode, i.e. they are not assembled in the store but are punched out in a special binary loader code and can be entered into the store by means of the SIR R.L.B. loader within 1-PASS SIR. The remaining options are not relevant.

11.3 Check mode.

When bit 64 has the value 1 and bit 2 has the value 0 a program will be scanned for errors without actually being assembled.

The only option available in this mode is 'display labels'. The effect of requiring other options is undefined.

(i) 2 bit (Main mode indicator)

This bit must be zero

(ii) 1 bit (Display labels)

This bit has the same effect as in the load-and-go mode.

11.4 Uses of non load-and-go assembly

Although it is usually more convenient to assemble programs in the load-and-go mode, non-load-and-go would be used in the following circumstances :-

(i) During the development of a program using a large number of proven subroutines or routines these subroutines and routines would be converted to RLB (intermediate) tapes, leaving only the new program itself to be loaded in telecode form. This saves time because RLB tapes are much smaller than SIR tapes and are read in at six times the speed.

Note that on completion the program and its subroutines and routines would be converted to pure binary by 2-PASS SIR.

(ii) As the means of incorporating code-procedures written in SIR to the 920 ALGOL System, described elsewhere.

11.5 Summary and Examples of Options

Mode			Effect
Load-and-Go	Translate to paper tape	Check	
2	0	64	Basic mode
1	-	1	Display Labels
4	-	-	Clear store
			:
16	-	-	Start placing program at 8
32	-	-	Set Dictionary below program

Add together the numbers in the appropriate column and precede the sum by an asterisk. e.g.

* 19	Load-and-Go, start placing program at location 8, display labels.
* 0	Translate to paper tape.
* 65	Checking mode, display labels.

12.1 Assembly of SIR Tapes.

The assembler "1-PASS SIR, 24/3/71, Binary Mode 3" is read in by the initial instructions, in Mode 3. All tapes written in SIR can then be read in by entering the assembler at one of the following starting addresses; also in Mode 3;

Address	Name	Effect
&17740 / &17741 OCTAL	START	Cancel all existing dictionaries and begin assembly, giving error indications and label lists in, respectively, 900-Series Telecode / 920 Telecode
&17742. OCTAL	CONTINUE	Assemble, maintaining current dictionaries

12.1.1 Load-and-Go Mode.

In this mode programs are assembled in the store ready for immediate running. During assembly appropriate error indications and, if required by the options, a label list are displayed.

When % is displayed the assembler locates literals and displays a list of unlocated identifiers followed by

FIRST a1 LAST a2

where a1 is the lowest and a2 is the highest address to which words have been placed since an entry was last made at START.

12.1.2 Non Load-and-Go Mode.

In this mode programs are output to paper tape in relocatable binary† (RLB) form. If required by the options they are preceded by a loader. The assembler forms and stores a checksum. When % is read this checksum is output followed by fifteen blanks and a loader halt code. Any necessary EU messages for global identifiers are then displayed. (EU messages are explained in Chapter 13). These are not necessarily errors, as the labels may be supplied by another relocatable binary tape. They must be distinguished from EU messages for missing local identifiers in the last block, which are displayed before the loader halt code. If any errors are detected during assembly, punching of the relocatable binary tape ceases and compilation continues in the Checking mode.

12.1.3 Checking Mode.

In this mode error indications and, if required by the options a label list are displayed. No other output occurs. The only store space used is that occupied by the dictionaries.

---

† or "intermediate" form

12. 2 Loading of Relocatable Binary Tapes.

RLB tapes can be entered into the store at one of the following starting addresses; in Mode 3 (but see paragraph 12.5)

Address	Name	Effect
&17734 &17735 OCTAL	START A	Cancel the current existing dictionary and read a relocatable binary tape. Start placing it at location 8 unless it begins with a PATCH to a different starting address, giving error indications and label lists in, respectively, {900-Series Telecode, 920 Telecode}
&17736	START B	Read a relocatable binary tape maintaining the current dictionary.
&17737	START C	As for ~~10~~ START A, but CPA is not reset to 8.

Once this has been done it is not possible to assemble source tapes without reading in the assembler again.

During loading, a list of global labels used and their addresses is displayed. If any errors are detected an error indication is displayed and the loader halts, but loading may be continued by entering at START B to find further errors. The effect of an attempt to run such a program is undefined. On reading a loader stop code loading stops, the loader displays a list of global identifiers still to be located preceding each identifier by 'FU'. It then displays a FIRST LAST message as described in 12. 1. 1 above with a1 referring to the last entry at START A or START C. The checksum preceding a loader stop code is checked against the checksum the loader has made whilst loading.

Every RLB tape must have a loader stop code at its end (i. e. the last source tape used in its production must end with N % N H).

12.3 Mixing of RLB tapes and mnemonic tapes.

It is possible to read several mnemonic tapes into the store using the assembler, and then to read several RLB tapes in at START B using the loader in the assembler. In this case all the tapes will share the same dictionary and can communicate with each other via global identifiers. This facility permits library subroutines to be stored as RLB tapes and a SIR program to use them without itself having to be translated to RLB form. Note that the last of the mnemonic tapes must end with new line % new line, (H).

12.4 Loading programs into the high end of the store.

Programs read in load-and-go are entered into the store immediately above tha last program read in, unless the 16 bit in the options indicates that the program is to be stored in location 8 onwards. Programs can, if necessary, be directed into a specified part of the store either by means of a patch at the start of the program or by use of the 'continue at 8' option followed by a skip, preferably the latter, to avoid trouble with any '$' symbols.

12.5. Compatability with earlier issues of 1-PASS SIR.

R.L.B. tapes made using "1-PASS SIR 2/6/66" may be loaded as in section 12.2. above, but using tape reader mode 1. (R.L.B. tapes made by the 2/6/66 and 24/3/71 issues of 1-PASS SIR, made from source-tapes not containing alphanumeric groups, are identical except for their mode of input.)

12.6 Multiple Program Assembly

If two or more programs are to be used together, linked by common global identifiers, and each program is terminated by %; the following rules should be observed.

12.6.1 Load and Go Assembly

Assemble the first tape by entry at START. This tape may have any load-and-go option. Assemble all subsequent tapes by entry at CONTINUE. These tapes may include any load-and-go option (except options including the 32 bit).

12.6.2 Assembly to paper tape (non-load-and-go)

(1) Assemble the first tape by entry at START. The first significant item on this tape must be the option * 0 No other options can occur on this tape.

(2) If the program is continued on further tapes, assemble these by entry at CONTINUE, until the % is reached. There must not be any options on these tapes.

(3) To assemble the succeeding linked programs steps (1) and (2) must be repeated for each program.

(4) When loading the programs the first program may be loaded by entry at STARTA or STARTC.

(5) The succeeding programs must be loaded by entry to STARTB.

Note: 1-PASS SIR will STOP if the first character on a source-tape is NOT a "newline"

Chapter 13: ERROR INDICATIONS

Error indications given during assembly.

The following error indications are displayed (i.e. output to the teleprinter) during the assembly of SIR tapes whenever the appropriate error is detected:-

Error	Meaning	Effect in Load-and-Go Mode
EO:	Instruction Error (i) function > 15 (ii) address part of quasi-instruction not absolute.	One store location is left unfilled.
E1:	Contextual Error Any impermissible sequence of characters not giving any other error indication	One store location is left unfilled.
E2:	Octal or Alphanumeric Error (i) Too many characters in an octal or alphanumeric group. (ii) character in octal group other than digits 0 - 7.	One store location is left unfilled.
E3:	Label declared Twice Label found identical to a previous label in block where previous label is still valid.	One store location is left unfilled.
E4:	Global Identifier not Beginning with Letter Applies only to identifiers in a Global Identifier List.	The program is corrupted in an undefined manner.
E5:	Store Full Program is about to over-write dictionary, or vice-versa. (This may be the result of a Patch error). (E5 after % has been read means that there is insufficient room to locate all the literals used in the program.)	The Compiler waits. Compilation can be continued. A patch, skip, option or obeyed instruction must be read next.

Error	Meaning	Effect in Load-and-Go Mode
E6:	Number Overflow (i) integer outside range -131, 071 to +131, 071 (ii)more than six digits in fraction.	One store location is left unfilled
E7:	Buffer Overflow Over 120 characters in line of text (i. e. too many for read buffer).	One store location is left unfilled.
E8:	Illegal Character (i) Misread or mispunched tape. (ii)character on tape having no internal code value. (iii) Parity Error	One store location is left unfilled.
E9:	Stop Code not first Character on Line Characters other than blanks or erases between 'new line' and stop code.	The Compiler waits. Compilation can be continued. One store location is left unfilled.
EG:	Global Label Error An attempt has been made to redefine a global label as sub-global.	Compilation continues.
EL:	Literal Error A literal has been used with an instruction other than 0, 1, 2, 4, 6, 12 or 13.	One store location is left unfilled.
EP:	Patch Error A patch, or obeyed instruction, refers to an unlocated address.	The Compiler waits. Compilation can be continued A patch skip, option or obeyed instruction must be read next.
EU:	Unlocated Identifier Identifier has appeared but never as a label. Given at end of block for local identifiers, or on reading new line % new line for global or sub-global identifiers.	Compilation continues

13.1 Layout of Error Indications and Their Effect on Assembly.

Each error indication is preceded by 10 "erase" characters. Three different types of layout are used for assembly error indications:-

(i) † EU: EU is displayed on a new line, followed by the identifier which has been detected as unlocated and an 'address'. If this 'address' is 8191 the identifier appears only in Global label lists, otherwise it is the address of the last reference to the identifier. The assembler continues checking the identifiers in the dictionary.

(ii) * E5, E9 and EP: E5, E9 or EP is displayed on a new line followed by the bracket count (i.e. the number of '['s found since the last START). Assembly is halted but it may be restarted at CONTINUE.

(iii) En (all others): En is displayed on a new line followed by the bracket count, and on the next line is displayed the line of source text in which the error was detected. The assembly continues with the examination of the next line of text.

In all cases, output of relocatable binary tape ceases if assembly is to paper tape, but error indications (and labels if requested) continue to be displayed.

13.2 Examples of Assembly Error Indications.

E2	16		8 occurs in an octal group in block 16
PRINT	6	&800000	
EO	10		Missing separator giving rise to an impossible function in block 10.
	152048		

---

† Note: EU displayed after % has been read is not necessarily an error indication. It may mean that a Global label, which belongs to a program that has not yet been loaded, has been referred to.

* Note: 2-PASS SIR prints a line of source text on finding E5 or E9; part or all of this line may be irrelevant or rubbish.

13.3 Error Indications given during loading of relocatable binary tapes.

The following error indications may be given during the loading relocatable binary tapes:-

Error Indication	Meaning
FA): Mis-read or FD): mispunched tape	two different kinds of illegal codes on RLB tape
FC: Label used twice	as for E3
FE: Store overflow	as for E5
FF: Checksum failure	punched checksum does not equal checksum added by loader.
FP: Unallocated address error	as for EP
FU: Unallocated label	as for EU

Note that:

(i) FC is displayed when a tape with a label in it is entered at START B when the same label has already occurred in a previous tape of the same program (the presence of two identical labels on one tape would have already been detected as an error during assembly).

(ii) FU indications will be displayed when a global identifier occurs in one tape and refers to a label on another tape which has not yet been entered. FU indications only indicate errors, therefore, if they are given after all the tapes of a program have been read in.

(iii) Since all local & sub-global identifiers are eliminated during assembly of the RLB tape FC and FU refer to global identifiers.

No additional information is displayed for F errors except that for FU errors the identifier which is unlocated is displayed on the same line as the FU.* All F errors halt the loader, but loading may be recontinued at START B.

---

*and an address is given corresponding to the address in EU errors.

Chapter 14: EXAMPLE OF A SIR PROGRAM

The following short program adds up the absolute values of the ten integers in the block headed 'DATA' and stores the answer in location ANSWER. *If, however, the sum becomes too large to hold in one* store location the letters OF are punched out on a new line and /15 8191* is put in location ANSWER. The program tape is read in first at START and it will stop on the stop code. The data block following the stop code, which can be on a separate tape if desired, is then read in at CONTINUE. The program can then be triggered at location BEGIN, the absolute address identified by being read off the label list which is produced as shown below:-

TT	BEGIN	8
LOOP	12	
OF	25	
END	32	
COUNT	34	
SUM	35	
T	ANSWER	36
TT	DATA	37
FIRST	LAST	
8	52	

The block DATA occupies locations 37 to 46 and the literals occupy locations 47 to 52: the first literal being placed in the lowest address.

---

* −1 would be less pathalogical.

*23

((SIR PROGRAM EXAMPLE)

```
[BEGIN DATA ANSWER]
BEGIN       4     -10          (ENTRY ADDRESS)
            5     COUNT
            4     +0
            5     SUM

LOOP        0     COUNT
            /4    DATA+10
            9     ;+2
            8     ;+2
            2     +0
            1     SUM
            5     SUM
            9     OF
            10    COUNT
            4     COUNT
            9     LOOP
            4     SUM
            8     END

OF          4     &022         (PUNCH NEW LINE)
            15    6144
            4     &137         (PUNCH O)
            15    6144
            4     &126         (PUNCH F)
            15    6144
            4     =/15 8191
END         5     ANSWER
            8     ;+0
COUNT       >1
SUM         >1
ANSWER      >1
(HALT CODE)
[DATA]
DATA        +65
            +12
            -14
            -756
            +602
            -5
            +56
            +1
            +0
            -22

%
```

14.1 Notes

(i) option * 23 means load and go, list labels, clear the store and start assembly at 8

(ii) relative addresses have been used for short jumps and identified addresses for longer jumps.

(iii) the identifiers here perform several different roles - LOOP, END and OF denote locations to be jumped to.
COUNT and SUM denote workspace
ANSWER denotes a location holding the result
BEGIN identifies the trigger address on the label list.

(iv) the octal values, with parity,* of the characters to be punched have been used; in a long program this would be done using alphanumeric groups together with a code table and print routine.

(v) the program occupies locations 8 to 46 and the six literals used (-10, +0, &022, &127, &126 and =/15 8191 occupy locations 47 to 52. The location given under LAST in the print-up is therefore 52.

(vi) the halt code at the end of the first block is on the line following the comment (HALT CODE).

(vii) % is preceded and followed by 'new line'.

(viii) BEGIN and ANSWER have been declared as Global labels so that other programs can refer to them. DATA is not wanted outside the program and has consequently been declared as Sub-Global.

14.2 Layout.

As separators can be inserted at will between the elements of a SIR program, considerable variety of layout is possible. It is recommended, however, that the layout used in the example be adopted. Note that extra 'new lines' may be used to break the print up into convenient portions.

---

*In 920 Telecode

CHAPTER 15: SUMMARY OF 1-PASS SIR ENTRY POINTS

Octal Entry Point	Action			Reference
&17734	STARTA	900	To load R.L.B tapes	12.2
&17735	STARTA	920		
&17736	STARTB			
&17737	STARTC			
&17740	START	900	To load Source tapes	12.1
&17741	START	920		
&17742	CONTINUE			
&17743	EXECUTE stored program			10.2

CHAPTER 16: STORE USED BY 1-PASS SIR

"1-PASS SIR, 24/3/71, Binary Mode 3" occupies locations 5620–8166. When assembling a program the dictionary occupies the store below 5620, growing towards 8, unless option bit 32 is used.

The part of 1-PASS SIR forming the R.L.B. loader occupy locations 7000–8166. When assembling a program entirely from R.L.B. tapes the dictionary occupies the store below 7000.

Locations 8167–8179 are NOT used by 1-PASS SIR; this permits other binary tapes in "A.C.D. 900-Series 18-bit Binary Tape Format 1/4/~~70~~" (described elsewhere) to be read into locations 2–56~~20~~19 without corrupting SIR.

17.1. General Description.

This version of SIR has been written for the following reasons :-

(i) To enable a SIR program to be written which occupies virtually an entire 8192-word store. Programs assembled by 1-PASS SIR are limited to about 4000 to 5000 words in load-&-go mode, and about 7000 words in RLB mode. Programs assembled by 2-PASS SIR may occupy all locations from 2 to 8166, inclusive.

(ii) To enable SIR programs to be written for use in a 16384-word store.

(iii) To produce a self-contained binary tape of any completed program; capable of reloading at the speed of the reader.

Any SIR program not containing options or obeyed instructions punched for 1-PASS SIR is accepted by 2-PASS SIR, BUT EVERY TAPE must start with a "newline". (See paragraph 1.4). Sub-global identifiers are not distinguished from global identifiers.

Patches can take the following forms:-

(a) ↑N where N is an integer in the range $1 \leq N \leq 16383$. This sets the store pointer to N.

(b) ↑"N or ↑~N where N is an integer in the range $0 \leq N \leq 8191$. This sets the store pointer to 8192+N.

(c) ↑LABEL or ↑LABEL±N where N is an integer and LABEL is a previously located label, either a local label of the current block or a global label "currently available".

17.2. Trigger Facility.

2-PASS SIR contains a trigger facility. If the symbol "<" occurs in a SIR program (preceded and followed by at least one newline, space, or tab) the binary tape made of this program will trigger, automatically, when the tape is loaded, at the current value of the Store Pointer or Current Placing Address, provided that this is in the range 2 to 8166 inclusive, and provided that the binary tape sum-check succeeds.

For example, the binary tape of the program

```
START   <   4  +0
           15  6144
            8  ;+0
```

will automatically start at START when loaded. The same effect can be achieved in retrospect by means of a patch, for example

```
↑START
<  $
```

(A patch of this type will be needed if a program is being assembled onto more than one tape by means of option bit 32, (see paragraph 17.3), if the first instruction to be obeyed is NOT contained in the last binary tape; since the trigger is punched into the tail of the tape being punched when the "<" symbol is read.)

Note that the instructions

```
↑2  <   4  ;+3
        5  8177
        8  ;+0
        8  START
```

may be used to set up an AUTOSTART, and that

```
↑2  <   4  ;+2
        5  8177
        8  START
```

will set up an autostart AND trigger the program when loaded. (In both cases a "$" or another patch should follow) In multi-level programs, these instructions will be corrupted by the S.C.R.'s & B-registers of the program being entered at START; but this is of no consequence.

17.3. Options.

In 2-PASS SIR the separate bits in an option have the following meanings:

1.	list labels - decimal addresses
2.	list labels - octal addresses
4.	punch zeros for skips
8.	set store pointer to 8192
16.	set store pointer to 8
32.	tie off present binary tape with a sumcheck; punch 360 blanks and then punch loader and store pointer for new binary tape.

In addition, a clear-store may be punched at the start of the binary tape by adding the size of the store to the option; i.e. if 8192 is added to the option a clear store will be punched for locations 2 to 8175, if 16384 or any larger multiple of 8192 is added to the option, two clear stores will be punched, the first to clear locations 8192 upwards, and the second to clear 2 to 8175. Clearly this facility can only be used in options which precede the first word of the program.

Option bit 4, and the clear-store bits, should NOT be required when assembling a correctly written SIR program.

Option bit 32 enables a long program to be assembled as several binary tapes.

Although bits 4, 32, and the clear-store bits have no meaning on the first pass, and bits 1 and 2 have no meaning on the second pass, the SAME options must be used on both passes (otherwise the sum-check will fail).

17.4. Literals.

The locating of literals is similar to 1-PASS SIR - i.e. though it is often not necessary to consider where they are to be located, they can be placed in any block of consecutive locations at will by means of patches. Literals for the two stores are considered independently. Those for the lower store will be located immediately following the last location used by the lower store, and those for the upper store will be located immediately following the last location used by the upper store.

Programs are allowed to jump from store to store indefinitely. For example, the following program would be acceptable:-

			locations
*23			
LABEL	4	+123	8
	5	A	9
	6	&306	10
↑163			
	4	-1	163
A		+0	164
↑12345			
	6	+1	12345
↑"100			
	7	;+0	8292
↑LABEL+7			
	4	-1	15
*15			
	3	AR	8192
	4	+1	8193
	6	-1	8194
	8	;+0	8195
%			

The literals used in the lower store and upper store are treated independently. In the example, the literals for the lower store are +123 &306, -1 and the literals for the upper store are +1, -1.

The lower store literals will be located in locations 16, 17, 18 (because 15 is the last location used in the lower store) and the upper store literals will be located in locations 8196, 8197 (because 8195 is the last location used in the upper store). Please note that the upper store literals are not located in the order they are written.

17.5. Assembly & Loading, & the label list.

The tape "2-PASS SIR, 7/1/71, Binary Mode 3" is read in by initial instructions in Mode 3. The SIR Telecode tapes, in 900-Series Telecode or 920 Telecode, are read in Mode 3. The first tape is entered at 8 or 12, and all the others at 9.

During the first pass the assembler checks for errors and stores the dictionary. Titles, error indications (if any) & addresses of labels (if option bits 1 or 2 are present) are output on the punch: in 920 Telecode if the first tape was entered at 8, and in 900-Series Telecode if the first tape was entered at 12. Local labels are preceded by 2 spaces. Reading will only stop if a halt-code is read or if a 'store-full' or 'patch' error (E5) is encountered.

When a '%' is read, a message of the form

FIRST	NEXT1	NEXT2	DICT
8	5432	12345	30%

will be punched, where FIRST is the lowest location used by the program, NEXT1 and NEXT2 are the next available locations, after the literals, in the lower and upper store modules (not necessarily the highest locations used by the program: thus FIRST has the same meaning as in 1-PASS SIR 2/6/66, but NEXT1 may not be the same as LAST of 1-PASS SIR, plus one, although it often is.) DICT indicates the percentage of available dictionary workspace used to assemble the program

The second pass is made by entering the first tape at 10 and all others at 11. If errors were found on the first pass the assembler stops, when entered at 10, with the number of errors in the accumulator. Entry 13 may be used to assemble in spite of these errors.

During the second pass, the binary tape or tapes are produced, the number of tapes being one greater than the number of options containing bit 32.

17.6. Dumping the dictionary & assembling frigs.

The Global dictionary of a program, the associated dictionary pointers, and a copy of the assembler may be dumped as a binary tape by entering at 16. This tape is useful for subsequent frigs.

Suppose a large program has been compiled onto binary tape by 2-PASS SIR, placed in the store, and a mistake is found in it. If there is no room in store for the 1-PASS SIR assembler, or AMEND, or similar, a correction frig can be translated to binary using the dumped dictionary.

The frig may consist of any number of blocks; patches and instructions in it may refer to global labels located in the main program. If a patch occurs before the first global identifier list it may refer to ANY global of the main program, but patches after the first global list of the frig may only refer to globals if they appear in the preceding global list. It appears to be necessary to start a frig with an option: otherwise 'ES' is given.

To compile a frig, place the dump of the dictionary and assembler into the store (preferably of another machine) by means of initial instructions. The first pass of the frig is entered at 14 (in which case the literals will be located as usual; i.e. after the final values of the store pointers of the frig) or at 15 (in which case the literals will be located after the literals of the main program (OR previous frig assembled). Enter the frig again at 10 in order to obtain a binary tape.

Label lists punched whilst assembling frigs will be in the same Telecode as those of the main tape.

17.7. Summary of Error Indications.

Basically the indications are the same as when 1-PASS SIR is used in the load-&-go mode, they are preceded by 10 "erase" characters, and "tab" is punched as "space".
Some of the individual meanings are slightly different:–

(i) E1 is given if the symbol "<" is used when $SP \geq 8192$.

(ii) E5 can mean any of the following:

(a) Unlocated identifier in a labelled patch (e. g. ↑PASS + 2). This will always occur if, for example, a tape consisting of such things is entered at 8 (which always effectively destroys the present dictionary) instead of 9, 14 or 15.

(b) Dictionary Store full.

(c) The store pointer falls outside one of the following ranges:

$$1 \leqslant SP \leqslant 8166$$
$$8192 \leqslant SP \leqslant 16383$$

(d) A skip has caused a transition from one store to the other. The only means of doing this is by using a patch or option.

(iii) E8 will be given for parity errors.

(iv) E11 will be given when % is read, if the same global identifier appears in one global list more than once.

(v) 2-PASS SIR will STOP:

(a) If any tape does not start with a "newline"
(b) At the start of the second pass, if errors were found on the first pass (see paragraph 17.5)
(c) During the second pass, if the address of a label disagrees with its value on the first pass (probably caused by loading tapes in wrong order)
(d) When % is read on the second pass (i.e. before the literals have been punched) if the sum of all characters read on the first & second passes disagree.

17.8. Summary of Entry Points.

8. Read first tape for first pass, giving label lists or error indications in 920 Telecode.
9. Read further tapes for first pass.
10. Read first tape for second pass.
11. Read further tapes for second pass.
12. Read first tape for first pass, giving label lists or error indications in 900-Series Telecode.
13. Same as 10, but ignore errors found on first pass.
14. Same as 8 or 12, but retain dictionary.
15. Same as 8 or 12, but retain dictionary & locate literals after literals of main tape.
16. Dump Assembler + Dictionary.

17.9. Binary tapes punched by 2-PASS SIR.

The binary tapes punched by "2-PASS SIR, 7/1/71, Binary Mode 3" are in "A.C.D. 900-Series 18-Bit Binary Tape Format, 1/4/70" as described elsewhere.

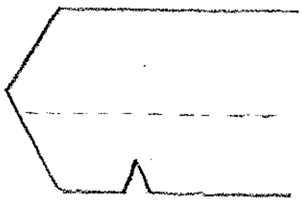
They are read into store by initial instructions using tape reader Mode 3. They are sum-checked; if the check fails, continuous output is given on the punch; if it succeeds then the program is triggered (if this facility has been used, see paragraph 17.2) or a dynamic stop is reached.

1- PASS SIR Options

2- PASS SIR Options

These two tapes each comprise 4 options and a % sign, each followed by a haltcode and a few blanks. The 4 options are the most useful ones for use with 1-PASS and 2-PASS SIR.

Since these tapes are short, receive much wear & are easily lost it is suggested that they are made on mylar tape and identified in the manner shown.

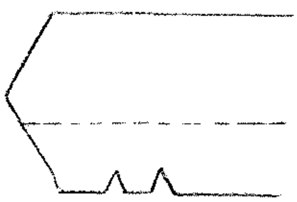


Useful options for 1-pass SIR

*23 Decimal & Octal Labels }
*22 No " } Clear store, assemble from 8. One of these options should be used at the start of assembly

*3 Decimal & Octal " }
*2 No " } These options may be used DURING assembly to start or stop labels.

%



Useful options for 2-pass SIR

*16 No Labels }
*19 Decimal & Octal " } skips for skips }

*20 No " }
*23 Decimal & Octal " } zeros for skips } Assemble from 8. One of these options should be used at the start of assembly.

%