ATLAS COMPILER COMPILER LISTING (1963)

This document explains the contents of a red spring-back binder, which contains a listing of the bootstrap loader of the Compiler Compiler printed on 22 December 1963. It was printed the day before Iain MacCallum left Ferranti/ICT for the Central Electricity Generating Board (CEGB). He had been working on the Compiler Compiler for his M.Sc. and on an Algol 60 compiler with John Clegg, under the direction of Dr Robin Kerr, for Ferranti/ICT. No Algol 60 definitions appear to have survived, perhaps because the first attempt at the compiler was much larger than had been hoped for, a consequence of the single-rooted structure of its syntax. It caused an unacceptable amount of drum swapping. The red binder was retained by Iain while he was at the CEGB and subsequently throughout his time at the Department of Computer Science, University of Essex. On retiring from the University in 2001, he passed the binder to Simon Lavington who took care of it. In October 2013, Simon asked Iain if he recognised its contents! It was the date on the listing of various test runs at the end that confirmed the authenticity of the document: it was printed on Iain's last two days with Ferranti/ICT!

By an astonishing coincidence, a second copy of the listing was preserved in the Department (now School) of Computer Science at Manchester University. It was probably gathered by Derrick Morris, and then passed to the late Dr Brian Napper, who took it upon himself to assemble a large box of material relating to the Compiler Compiler where it remained in the School's archive after Dr Napper's death. In mid 2013 it was borrowed by Dik Leatherdale by kind permission of Prof. Jim Miles, the current head of the School. The contents of the box included a card bearing the warning "THIS BOX INCLUDES PRICELESS ARCHIVE MATERIAL. DO NOT TOUCH WITHOUT Dr. R.B.E.NAPPER'S PERMISSION". It is thought that this second copy is a carbon copy of Iain MacCallum's as it is identical down to the page breaks which, in a few cases, run through the text of some lines. The box also contained the surviving copy of Jeff Rohl's flowcharts and notes on the Compiler Compiler as well as various listings which appear to relate to an attempt to transfer the Compiler Compiler to the Cambridge Atlas II (a.k.a. Titan). It is known that CC was used to create and support the Systems Assembly Language (SAL) at Cambridge – [see ref. 1].

The document in the red binder (and the carbon copy) presented a challenge to Iain MacCallum, one of the original CC implementors, and Dik Leatherdale, author of an Atlas 1 Emulator – [see ref. 3]: could they be scanned and used to reconstruct the Compiler Compiler on an emulator, running on a PC? By December 2013, Iain and Dik had managed to scan and load the entire bootstrap into the emulator, thus verifying the accuracy of all but a few routines at the end of the scan. There followed an extensive investigation by Dik and Bill Purvis (author of another Atlas 1 emulator) of the meaning of DEFINE COMPILER, and its relationship with the infant operating system of 1963. Certain changes were made to adapt the program to the later, mature operating system definition supported by the emulators. This was essential if the

reconstructed CC was ever to compile a compiler! In May 2014 they managed to define a trivial program written in the CC language and execute it correctly on the two different emulators.

The Original Compiler Compiler code, with line numbers and contemporary comments, is referred to in this document as 'cc source_1.pdf' – available as [ref. 5].

Structure of the listings

The listings in the red binder have been split into four sections, by content. The original input was, of course paper tape which has not survived. However, the CC Index (see file Appendix 4) refers to paper tapes B, C, D, E, F, G and H. It should, therefore be possible to identify these tapes on the listings. Sections 1 to 3 are the Compiler Compiler itself, and section 4 is the output from six test runs. (See file SixRuns.pdf – available as [ref. 6].)

Section 1

Section 1 (CC source_1.pdf lines 1 to 2143) consists of 30 foolscap pages of Atlas Octal Input, containing up to 75 lines per page, numbered by hand from 1 to 30. These pages were produced by a cross assembler for Atlas running on the Mercury. The original input to this cross assembler has not been found. These pages in the binder were printed from 5-hole paper tape on a Creed Teleprinter. It took a sensitive OCR scanner (Omnipage 18) and many hours of human intervention to retrieve the code from this listing. This section is read by the Octal Input routine in the Atlas Fixed Store.

From a brief glance at Section 2, (pages 2-1 to 2-107 - lines 2154 to 7954) which is effectively the data for the program formed in Section 1, it is easy to mistake Section 1 for a conventional assembler of Atlas Intermediate Code. However, with reference to the list of contents of this section in Appendix 1, it can be seen that it is better described as a basic Compiler Compiler Kernel. It starts with the index and the main chain store both of which are used throughout all phases of the Compiler Compiler. It has simplified versions of fundamental routines such as the recursive routine entry code (DOWN) and exit code (END), the top-level **MASTER** routine, the syntactic ANALYSIS ROUTINE and the LINE RECONSTRUCTION Routine. As the **ITEM** routine assembles items in Section 2, it maintains the **ITEM** index.

Section 2

Consists of 107 foolscap pages of Atlas (CC) Intermediate Code (pages 2-1 to 2-107 - lines 2154 to 7954) printed on a Flexowriter, from 7-hole paper tape. These are Phrase Definitions, Format Dictionaries and more Routines. This section scanned reasonably well, the foolscap page size being the main drawback. Many pages had to be scanned twice and 'stitched' together.

A list of the items of Section 2 is provided in Appendix 2. The objective of Section 2 is to develop a minimal compiler that will read the Phases, Formats and Routines of

Section 3 in the full Compiler Compiler Language. It introduces a **DELETE ITEM** routine (ITEM 145), a **REPLACE ITEM** routine (ITEM 164) which is used to replace basic versions of, amongst others, the ANALYSIS Routine (ITEM 215), the LINE RECONSTRUCTION Routine (ITEM 238) and the **MASTER** Routine (ITEM 214). This version of the ANALYSIS Routine has to parse the relatively high level language statements seen in the listings of Section 3. The replacement LINE RECONSTRUCTION Routine admits multiple overstrikes to represent characters such as ≠ and ≥.

The three major routines of the Compiler Compiler that process the high level elements of the definition of a compiler are in this section: the **PHRASE** routine (ITEM 218), the **FORMAT** routine (ITEM 220) and the **ROUTINE** routine (ITEM 221). The hard work of processing the individual statements of a routine is done in ITEM 253, simply called 'compile body of a routine'.

Towards the end of this section are a number of hand-coded built-in routines for frequently used statements in the routines that define the semantics of the target language. As an example, four of these used for basic arithmetic on numbers and addresses are:

```
ITEM 187        [AB] = [WORD] [SEP]
ITEM 188        [AB] = [WORD][OPERATOR][WORD] [SEP]
ITEM 189        ([ADDR]) = [WORD] [SEP]
ITEM 190        ([ADDR]) = [WORD][OPERATOR][WORD] [SEP]
```

These are re-entered in the CC language in Section 3 with the same item numbers:

```
FORMAT [BS]     =[AB] = [WORD][SEP], 187
FORMAT [BS]     =[AB] = [WORD][OPERATOR][WORD][SEP], 188
FORMAT [BS]     =([ADDR]) = [WORD][SEP], 189
FORMAT [BS]     =([ADDR]) = [WORD][OPERATOR][WORD][SEP], 190
```

and

```
ROUTINE (COMPILER)[BS] ≡ [AB/1] = [WORD][SEP]
...
ROUTINE (COMPILER)[BS] ≡ [AB] = [WORD/1][OPERATOR][WORD/2][SEP]
...
ROUTINE (COMPILER)[BS] ≡ ([ADDR]) = [WORD][SEP]
...
ROUTINE (COMPILER)[BS] ≡ ([ADDR]) = [WORD/1][OPERATOR][WORD/2][SEP]
...
```

For an explanation of the qualifier **(COMPILER)** see Section 3 below.

The handling of LISTS, NESTS (stacks) and other relatively addressed data structures is by Auxiliary Statements **[AS]** such as **ADD WORD TO LIST** and **ADD WORD TO NEST**.

These are hand-coded in this section, and like the built-in statements, are replaced in Section 3 by equivalent procedures written in the language of the Compiler Compiler.

Section 3

Pages 3-01 to 3-75 (lines 7958 to 10961) consists of 75 foolscap pages of **PHRASE** definitions, Built-in statements **[BS]**, Auxiliary Statements **[AS]** and Routines, all in the language of the Compiler Compiler. See Appendix 3 for a list of all items in this Section.

This Section of the listing begins with a BNF-like definition of the syntax of the basic **PHRASE**s of the Compiler Compiler language. It is followed by the **FORMAT**s of the Built-in Statements **[BS]**, seven new Master Phrases **[MP]**, and the Auxiliary Statements **[AS]**, and finally the **COMPILER** versions of the Built-in Statements.

These pages were well printed and at first appeared to scan well. Like pages in the previous section, some needed to be 'stitched'. However, there were other problems in making digital sense of this section. The character set included the Greek letters **α** and **β** and the full range of comparator operators **≠**, **≡**, **≢**, **≤**, and **≥**. The OCR scanner made numerous mistakes with brackets and other non-alphanumeric characters.

What is the purpose of the so-called **COMPILER** routines? For example, at line 8050, is the definition of the Auxiliary Statement for adding a WORD to a LIST or a NEST.

```
ROUTINE [AS] ≡ ADD [WORD] TO [LIST OR NEST][AB][SEP]
           β69 = [AB]
           β68 = [WORD]
           α1 = CATEGORY OF [LIST OR NEST]
           α1 = α 1 + 203
           CALL R α1
           [AB] = β69
           END
```

Then at line 8830 is the **COMPILER** version.

```
ROUTINE (COMPILER)[AS] ≡ ADD [WORD] TO [LIST OR NEST][AB][SEP]
           CALL [BS] COMPILER B69 = [AB]
           CALL [BS] COMPILER B68 = [WORD]
           A1 = CATEGORY OF [LIST OR NEST]
           A1 = A1 + 203
           PLANT 1102, 70, 76, A1 IN B88
           CALL [BS] COMPILER [AB] = B69
           END
```

When the routine at line 8050 is input, the 6 statements are analysed line by line, and assembled as analysis records for subsequent interpretation when it is called. Interpretation is slow. When the **COMPILER** version is input, the 6 statements are similarly analysed line by line but in this case native Atlas instructions are planted. When this version is executed it is fast!!

It is worth examining the **COMPILER** version at line 8830.

1. The first statement

   ```
   CALL [BS] COMPILER B69 = [AB]
   ```

   needs a **COMPILER** version for the statement

   ```
   B69 = [AB]
   ```

   This is the routine starting at line 8061 running to line 8168, namely

   ```
   ROUTINE (COMPILER) [BS] ≡ [AB/1] = [WORD][SEP]
   ```

   So the first statement is compiled as native Atlas code.

2. The second statement is compiled into native code in exactly the same way.

3. The third and fourth statements

   ```
   A1 = CATEGORY OF [LIST OR NEST]
   A1 = A1 + 203
   ```

   trade on the juxtaposition of Items 204 and 205 in the index.

ITEM 204	Add word to list
ITEM 205	Add word to nest

   and the PHRASE definition

   ```
   PHRASE [LIST OR NEST] = LIST, NEST
   ```

   at line 7976. **LIST** is Category 1 of **[LIST OR NEST]** and **NEST** is Category 2.

   Thus, they compute in variable **A1**, the appropriate item number, but plant nothing

4. The next instruction does the planting of native code.

   ```
   PLANT 1102, 70, 76, A1 IN B88
   ```

5. Finally,

   ```
   CALL [BS] COMPILER [AB] = B69
   ```

calls the **COMPILER** version of **[AB/1] = [WORD]** used to plant code to update the address of the **LIST OR NEST [AB]**.

Section 4 - Various outputs from loading Sections 1 to 3.

The final section, on continuous perforated line printer paper, with the sprocket holes removed, consists of the listing of six runs of the Compiler Compiler. (See file SixRuns.pdf, [ref. 6]. These are best identified by the date and time of printing.

22.12.63 11.11.23
This is a Catastrophic Fault listing of the non-zero B-lines and the stack. The error was in R142, the pre editing routine. The circumstances and the reason have not been ascertained.

22.12.63 11.20.47
This is another Catastrophic Fault listing of the non-zero B-lines and the stack. Again the error was in R142, the pre editing routine. The circumstances and the reason have not been ascertained.

We found that the next two execution listings were essential to the loading of the reconstructed Compiler Compiler on the Atlas I emulators. As each batch of Optically scanned pages was added to the reconstructed input file, it was possible to check the output of the emulator with the output of 22 December 1963. Each discrepancy was investigated until an error in the source was located and corrected.

22.12.63 11.41.42
This is the output from the CC as it loaded Parts 1, 2 and part 3 as far as line 9360 (Tape G)

22.12.63 13.11.05
This is the output from the CC as it loaded Part 3 from line 9364 to the end. (Tape H)

23.12.63 10.45.12
This appears to be the result of DEFINE COMPILER CC followed by a list of the B-lines.

23.12.63 10.45.51
This appears to be the result of DEFINE COMPILER CC2 followed by a list of the B-lines.

Philosophy of the Compiler Compiler bootstrap

The Compiler Compiler is 'complete' in the sense that it may be written in its own language. In other words, it consists of phrase definitions, format dictionaries, and routines whose instructions belong to an extended set of the built-in instructions and auxiliary statements. The general implication of this is that once the material for interpreting one of these primary statements has been loaded, it is possible to process

subsequent statements of that type written in the system language. This type of bootstrapping procedure has been adopted for the Compiler Compiler for three reasons.

1. It has been possible to write a considerable part of the system in a language which is particularly suited to its own requirements and which reduces the likelihood of errors in the coding.

2. It provides more positive evidence that the various parts of the system function properly; for example, if a phrase which has been processed by the phrase assembly routine is subsequently used by other parts of the program and is found to give the expected results then it is almost certain that the phrase has been assembled correctly.

3. Once the Compiler Compiler is working on Atlas, it will be possible to produce a compiler for another computer by providing the Atlas version with a set of primary assembly routines which will plant machine instructions for the other computer. By reading the compiler again, written entirely in the language of the system, a compiler for the other machine will be generated.

The Phrase Definitions, Format Dictionaries and Routines in Section 2 of the listings are hard to decipher without the help of the index of item numbers/ descriptions. See Appendix 4.

Addressing Philosophy

The fundamental addressing unit on Atlas was the 48-bit *word;* used to contain an instruction or, more significantly, a floating-point number. The top 21 bits of the (24-bit) address were used to specify a word. But the word could also be divided into two 24-bit halfwords and eight 6-bit characters. To address these smaller quantities, the remaining 3 bits of the address were used. In normal Atlas parlance, an address might be specified as a decimal number with an optional octal "fraction". Thus the least significant halfword of word 91 would be specified as 91.4, and the least significant character of the same as 91.7.

But the Compiler Compiler made little or no use of floating point numbers, dealing almost exclusively with halfwords. So, by contrast its addressing was specified in terms of **halfwords** with a decimal number in the source code mapping onto the top **22** bits of the address and the character addresses being specified as a 2-bit **binary** fraction. Thus in CC, the same addresses would be specified as 183 and 183.11.

Our modern emulator [see ref. 3] displays information in "normal" Atlas addressing mode and whereas the Compiler Compiler itself finds no difficulty in employing its unusual addressing convention, it can be confusing for the human reader.

Relative Addressing

Once the decision had been taken to allow basic routines to be replaced by routines with more functionality, or which plant native code, it followed that all store references had to be relative. The rich instruction format of Atlas made this possible. Local variables are addressed relative to the start of the routine and jumps are all relative to the address of the jump instruction. Routine replacement consisted of adding the replacement routine - possibly using the original to do so with interrupts inhibited - sliding up memory over the original routine, updating indexes restoring interrupts and continuing reading routines.

However, certain routines, for example 0 (the compiler entry point), 281 (slide compiler up a block), 279 (sub-routine entry - DOWN) cannot be allowed to move in memory and are designated as **FIXED ITEM**s.

Acknowledgements

During much of his two and a half years at Manchester University Iain sat at a desk in the drawing office of the Electrical Engineering Department's Dover Street Building next to Jeff Rohl. Iain will never forget the diligence with which Jeff documented the Compiler Compiler in the form of hand-written flowcharts. Jeff is no longer with us but fortunately these gems found their way into the big box that Brian Napper guarded, and then to Dik who scanned them. Thus we had photo copies to refer to as we battled with the less than perfect OCR output. Samples of these flowcharts are available at [ref. 7]. In the 1960s, computer memory was too precious to have it cluttered with comments! So we record our grateful thanks posthumously to Jeff and Brian!

Iain is also grateful to Tony Brooker who took him into the Compiler Compiler team in 1961, and to Derrick Morris who, in Tony's absence in the USA, led the CC development and supervised his MSc. Alas, Derrick is no longer with us.

We are also indebted to Bill Purvis who provided the OCR scan of the Index that will help serious readers winkle out more detail from the bootstrap listings.

References.

1. See: *Tony Brooker and the Atlas Compiler Compiler:*
http://curation.cs.manchester.ac.uk/atlas/docs/Brooker%20Atlas%20CC%20rev%20April%202016bb.pdf

2. The definitive description of the Compiler Compiler is in this paper:
R.A. Brooker, I.R.MacCallum, D. Morris, J.S. Rohl. *'The Compiler Compiler'*, Annual Review in Automatic Programming, Vol.3, 229-71, Pergamon Press, 1963.
Also available at: http://curation.cs.manchester.ac.uk/atlas/docs/ccPaperDL.pdf

3. Dik Leatherdale's Atlas emulator:
https://www.leatherdale.net/atlas.htm

4**. I.R.MacCallum.** M.Sc. Thesis, *'Some aspects of the Implementation of the Compiler Compiler'.* Manchester University Thesis No. 7476. Also available at:
http://curation.cs.manchester.ac.uk/atlas/docs/Some%20aspects%20of%20the%20implementation%20of%20the%20Compiler%20Compiler%20on%20Atlas.pdf

5. File cc source_1.pdf is available at:
http://curation.cs.manchester.ac.uk/atlas/docs/Compiler%20Compiler%20source%20code.pdf

6. File SixRuns.pdf is available at:
http://curation.cs.manchester.ac.uk/atlas/docs/Six%20CC%20test%20runs.pdf

7. Samples of five of the flowcharts are available at:
http://curation.cs.manchester.ac.uk/atlas/docs/Original%20Compiler%20Compiler%20flowcharts.pdf

Iain MacCallum
Dik Leatherdale

September 2014

APPENDIX 1

A list of all the items in Section 1 of the listings in the Red Binder in the order in which they appear.

Octal addresses of the first word		Item
20001010 to 20002050		index (Items 130 to 266)
20002260 to 20007634		Chain
20016360	ITEM 239	DOWN
20017100	ITEM 240	END sequence
20017220	ITEM 215	ANALYSIS ROUTINE
20022540	ITEM 266	ITEM ROUTINE
20026670	ITEM 252	split chain into 2 subchains
20027060	ITEM 130	[MP] = PHRASE, ITEM, END OF MESSAGE, FORMAT, FORMAT CLASS, DELETE ITEM, REPLACE ITEM
20027460	ITEM 261	Convert metasyntactical symbols
20030560	ITEM 238	LINE RECONSTRUCTION routine
20032460	ITEM 245	General unpacking routine
	ITEM 247	Line Image
20037260	ITEM 246	Standard Flexowriter Tab Settings
20040150	ITEM 222	DUAL routine
20041430	ITEM 160	[N] (as 149 but not in CID), used by 146
20041720	ITEM 150	Initial Entry Routine
20042020	ITEM 214	MASTER routine
E20000000		End of octal input. Enter at this address

APPENDIX 2

A list of all items in Section 2 of the listing in the Red Binder in the order in which they appear.

ITEM 145	DELETE ITEM routine
ITEM 131	[BS] dictionary
ITEM 132	[AS] dictionary
ITEM 133	[SS] dictionary
ITEM 129	Conventional list of 20 format classes
ITEM 169	List of dictionaries to be packed
ITEM 256	Double entry list of routine/compiling version serial numbers
ITEM 134	CID dictionary
ITEM 243	General Packing Routine for dictionaries
ITEM 228	Transfer dictionary to record store
ITEM 216	24-bit multiplication and division
ITEM 164	REPLACE ITEM routine
ITEM 155	Delete an item in the store
ITEM 175	Second entry to Delete item
ITEM 231	Transfer dictionary to chain store
REPLACE ITEM 130	[MP] =
ITEM 245	General unpacking routine
ITEM 219	Print B82 in Octal
ITEM 142	pre-editing routine
ITEM 258	Non-catastrophic fault routine
ITEM 248	Decimal printing routine
ITEM 257	CATASTROPHIC FAULT routine
REPLACE ITEM 215	ANALYSIS routine
REPLACE ITEM 261	Convert metasyntactical symbols
REPLACE ITEM 238	LINE RECONSTRUCTION routine
REPLACE ITEM 246	Standard flexowriter tab settings
ITEM 227	END OF MESSAGE routine
REPLACE ITEM 214	MASTER routine

END OF MESSAGE

REPLACE ITEM 161	Set Chain and Stack
REPLACE ITEM 227	END OF MESSAGE routine

END OF MESSAGE

REPLACE ITEM 142 Pre-editing routine

END OF MESSAGE

ITEM 218	PHRASE routine
ITEM 260	Entry to PHRASE routine used by auxiliary phrase routine
ITEM 242	Auxiliary Phrase routine
ITEM 220	FORMAT routine

ITEM 213 Read next section
ITEM 141 Look up or enter in double-entry list
ITEM 259 Add nil branch to dictonary
ITEM 204 Add word to list
ITEM 205 Add word to nest
ITEM 206 Withdraw word from nest
ITEM 207 Delete chain
ITEM 209 Add list to list
ITEM 210 Copy linear list to chain
ITEM 171 [general phrase identifier] = [[phrase identifier][phrase label][phrase index]]
ITEM 174 [phrase label] = / [N], NIL
ITEM 176 [phrase index] = ([ABN]), NIL
ITEM 151 [body of phrase definition] = [143][π] = [phrase*]
ITEM 156 [phrase*] = [phrase][phrase*], [phrase] EOS
ITEM 157 [phrase] = BUT NOT, [π or ES*] COMMA, NIL, [π or ES*]
ITEM 159 [serial number] = COMMA [N], COMMA [π or ES*], NIL
ITEM 163 [body of format defn.] = [π]=[π or ES*][serial number]EOS
ITEM 139 skeleton of A* and A?

A* ≡ A* = A A* COMMA A EOS

A? ≡ A? = A COMMA NIL EOS

ITEM 143 (CR), NIL
ITEM 265 double-entry list for serial no of routine / compiling version
ITEM 225 merge new entry into dictionary
ITEM 224 General dictionary routine
ITEM 230 [identifier] conversion routine
ITEM 172 [phrase identifier] built-in
ITEM 158 Built-in phrase for any sequence of identifiers or basic symbols
ITEM 152 Built-in phrase for phrase identifier
ITEM 160 [N] (not in CID)
ITEM 149 Built in phrase for [N]
ITEM 166 Built-in phrase for [α] and [A]
ITEM 167 Built-in phrase for [β] and [B]
ITEM 229 COMMA
ITEM 183 Print Symbol
ITEM 177 Print new format or phrase

END OF MESSAGE

ITEM 278 FIXED ITEM routine
ITEM 253 compile body of a routine
ITEM 221 ROUTINE routine
ITEM 217 Convert absolute pointers to relative pointers
ITEM 223 'Is it parameter-free?' routine
ITEM 144 Phrase: [label] = [separator*?] EOS, [separator*?][primary label]
ITEM 185 [RESOLVED-P] = [set p][152][reset p] 1, [reset p] -1
ITEM 179 [SEP] = COMMA, EOL
ITEM 184 [LABEL]
ITEM 148 [separator*?] = COMMA, EOL
ITEM 181 [PI] denotes identifier / label?/index?
ITEM 250 reset p

ITEM 244 set p’
ITEM 251 [FD]
ITEM 173 [OW]

END OF MESSAGE

ITEM 187 [AB] = [WORD] [SEP]
ITEM 188 [AB] = [WORD][OPERATOR][WORD] [SEP]
ITEM 189 ([ADDR]) = [WORD] [SEP]
ITEM 190 ([ADDR]) = [WORD][OPERATOR][WORD] [SEP]
ITEM 191 [JUMP] [LABEL] [IU] [QI] [EQV] [RESOLVED-P] [SEP]
ITEM 192 [JUMP] [LABEL] [SEP]
ITEM 235 compute value of a word
ITEM 236 compute value of an address
ITEM 211 CALL R [PI] [SEP]
ITEM 197 [AB] = CATEGORY OF [PI] [SEP]
ITEM 198 [AB] = NUMBER OF [PI] [SEP]
ITEM 202 [AB] = ADDRESS OF [PI] [SEP]
ITEM 193 LET [PI] = [GENERATED-P] [SEP]
ITEM 194 [JUMP][LABEL][IU][PI][EQV][RESOLVED-P] [SEP]
ITEM 195 LET [PI][EQV][RESOLVED -P][SEP]
ITEM 196 [JUMP][LABEL][IU][PI] = [PI][SEP]
ITEM 234 look-up ([PI]) L.S.E. routine
ITEM 232 Transplant routine
ITEM 233 look-up
ITEM 170 interpret -> B82
ITEM 267 SHIFT [AB] UP [ABN] [SEP]
ITEM 268 SHIFT [AB] DOWN [ABN] [SEP]
ITEM 269 SPACE [SEP]
ITEM 270 NEWLINE [SEP]
ITEM 271 PRINT [ABN] [SEP]
ITEM 199 [FD][COMMA][WORD][COMMA][WORD][COMMA][WORD][SEP]
ITEM 200 PLANT [FD][COMMA][ABN][COMMA][ABN][COMMA][WORD]IN[B][SEP]
ITEM 203 [AB] = CLASS OF [PI] [SEP]
ITEM 201 [PI] = [AB] [SEP]
ITEM 208 CALL R [ABN] [SEP]
ITEM 212 [FD][COMMA][WORD][COMMA] 0 [COMMA]L[LABEL][SEP]
ITEM 262 END (i.e. the [BI] format)
ITEM 263 [AB] = INDEX [ABN] [SEP]
ITEM 264 INDEX [ABN] = [AB] [SEP]
ITEM 273 PRINT SYMBOL [ABN] [SEP]
ITEM 277 PRINT [ABN] IN OCTAL [SEP]

END OF MESSAGE

APPENDIX 3

A list of all items in Section 3 of the listing in the Red Binder in the order in which they appear.

```
PHRASE      [146] = [160]), NIL
PHRASE      [147] = [BS], [182][AS], [182][SS]
PHRASE (CR) [182] = *, NIL
PHRASE      [AB] = [A], [B]
PHRASE      [ABN] = [A], [B], [N]
PHRASE      [249] = (COMPILER), NIL
PHRASE      [254] = SMALL R [160], R[160], [249][181][EQV][185]
PHRASE      [186] = [152]
PHRASE (CR) [JUMP] = ->, >, JUMP, [83]
PHRASE      [OPERATOR] = +, -, X, /, &, V, #, AND, NOT EQV
PHRASE      [COMPARATOR] = =, #, >, ≤, <, ≥, )
PHRASE      [ADDR] = [AB] + [ABN], [AB]-[ABN], [AB] (+) [ABN], [AB]
PHRASE      [WORD] = [ADDR],([ADDR]),[-?][N].[0-3],[-?].[0-3],[-?][N],[173]
PHRASE      [-] = -
PHRASE      [IU] = IF, UNLESS
PHRASE (CR) [EQV] = ≡, (=)
PHRASE      [0-3] = 00, 01, 10, 11
PHRASE      [,WORD] = [COMMA] [WORD]
PHRASE      [LIST OR NEST] = LIST , NEST
```

```
FORMAT [BS] =[AB] = [WORD][SEP], 187
FORMAT [BS] =[AB] = [WORD][OPERATOR][WORD][SEP], 188
FORMAT [BS] =[AB] = ADDRESS OF [PI][SEP], 202
FORMAT [BS] =[AB] = CATEGORY OF [PI][SEP], 197
FORMAT [BS] =[AB] = CLASS OF [PI][SEP], 203
FORMAT [BS] =[AB] = NUMBER OF [PI][SEP], 198
FORMAT [BS] =[AB] = INDEX [ABN][SEP], 263
FORMAT [BS] =([ADDR]) = [WORD][SEP], 189
FORMAT [BS] =([ADDR]) = [WORD][OPERATOR][WORD][SEP], 190
FORMAT [BS] =LET [PI][EQV][RESOLVED -P][SEP], 195
FORMAT [BS] =LET [PI] = [GENERATED - P][SEP], 193
FORMAT [BS] =[JUMP][LABEL][SEP], 192
FORMAT [BS] =[JUMP][LABEL][IU][WORD][COMPARATOR][WORD][SEP], 191
FORMAT [BS] =[JUMP][LABEL][IU][PI][EQV][RESOLVED - P][SEP], 194
FORMAT [BS] =[JUMP][LABEL][IU][PI] = [PI][SEP], 196
FORMAT [BS] =PRINT [ABN] IN OCTAL [SEP] , 277
FORMAT [BS] =PRINT [ABN][SEP], 271
FORMAT [BS] =PRINT SYMBOL [ABN][SEP],273
FORMAT [BS] =PLANT [FD][COMMA][ABN][COMMA][ABN][COMMA][WORD]IN[B][SEP],200
FORMAT [BS] =[FD][COMMA][WORD][COMMA][WORD][COMMA][WORD][SEP], 199
FORMAT [BS] =[FD][COMMA][WORD][COMMA] 0 [COMMA]L[LABEL][SEP], 212
FORMAT [BS] =[PI]= [AB][SEP], 201
FORMAT [BS] =CALL R [ABN][SEP], 208
FORMAT [BS] =CALL R [PI][SEP], 211
FORMAT [BS] =END[SEP], 262
FORMAT [BS] =INDEX [ABN] = [AB][SEP], 264
FORMAT [BS] =SHIFT [AB] UP [ABN][SEP], 267
FORMAT [BS] =SHIFT [AB] DOWN [ABN][SEP], 268
FORMAT [BS] =SPACE [SEP], 269
FORMAT [BS] =NEWLINE [SEP], 270
FORMAT [BS] =[WORD]/[WORD][SEP]
```

```
FORMAT [MP] = END OF PRIMARY MATERIAL, 226
FORMAT [MP] = ROUTINE, 221
FORMAT [MP] = BUILT-IN PHRASE, 292
FORMAT [MP] = DEFINE COMPILER, 275
FORMAT [MP] = AMEND COMPILER, 276
FORMAT [MP] = FIXED ITEM , 278
FORMAT [MP] = SLIDE COMPILER UP A BLOCK , 281



FORMAT [AS] =     [AB] = [LIST OR NEST][WORD][SEP]
FORMAT [AS] =     [AB] = [LIST OR NEST]([WORD][, WORD*])[SEP]
FORMAT [AS] =     [AB] = LIST [PI][SEP]
FORMAT [AS] =     [AB] = LIST [AB]([ABN][COMMA]?)[SEP]
FORMAT [AS] =     [AB] = VALUE OF LIST [AB] IN DICT [AB][SEP]
FORMAT [AS] =     [AB] = CONVENTIONAL LIST OF [ABN] WORDS [SEP]
FORMAT [AS] =     DELETE CONVENTIONAL LIST [AB][SEP]
FORMAT [AS] =     DELETE [LIST OR NEST] [AB][SEP]
FORMAT [AS] =     DELETE LIST [AB] FROM DICT [AB][SEP]
FORMAT [AS] =     ADD ([WORD][, WORD*]) TO [LIST OR NEST][AB][SEP]
FORMAT [AS] =     ADD [WORD] TO [LIST OR NEST][AB][SEP]
FORMAT [AS] =     ADD LIST [AB][COMMA][WORD] TO DICT [AB][SEP]
FORMAT [AS] =     ASSIGN VALUE [ABN] TO [PI][SEP]
FORMAT [AS] =     ANALYSE LIST [AB] W.R.T. [PI][SEP]
FORMAT [AS] =     WITHDRAW [AB] FROM NEST [AB][SEP]
FORMAT [AS] =     LIST [AB] = LIST [AB] + LIST [AB][SEP]
FORMAT [AS] =     LIST [AB] = ENTRY WITH VALUE [AB] IN DICT [AB][SEP]
FORMAT [AS] =     LIST [AB] = NEXT LINE FROM INPUT [N][SEP]
FORMAT [AS] =     LIST [AB] = NEXT RECONSTRUCTED LINE [SEP]
FORMAT [AS] =     CONVERT [PI] TO [AB][SEP]
FORMAT [AS] =     CALL [PI] COMPILER [GENERATED-P]
FORMAT [AS] =     MONITOR ([ALL SYMBOLS EXCEPT RT BRACKET])[SEP]
FORMAT [AS] =     PRINT LIST [ABN][SEP]

ROUTINE [AS] ≡    ADD [WORD] TO [LIST OR NEST][AB][SEP]
ROUTINE (COMPILER) [BS] ≡ [AB/1] = [WORD][SEP]
ROUTINE [AS]  ≡   CALL[PI] COMPILER [GENERATED-P]
ROUTINE(COMPILER)[BS] ≡ ([ADDR]) = [WORD][SEP]
ROUTINE (COMPILER) [BS] ≡ PLANT
[FD][COMMA][ABN/1][COMMA][ABN/2][COMMA][WORD]IN[B][SEP]
ROUTINE (COMPILER)[BS] ≡ ->[LABEL][IU][WORD/1][COMPARATOR][WORD/2][SEP]

ROUTINE (COMPILER)[BS] ≡      [AB] = [WORD/1][OPERATOR][WORD/2][SEP]
ROUTINE (COMPILER)[AS] ≡ ADD [WORD] TO [LIST OR NEST][AB][SEP]
ROUTINE (COMPILER) [BS] ≡ [AB] = CATEGORY OF [PI] [SEP]
ROUTINE (COMPILER) [BS] ≡ [AB] = ADDRESS OF [PI] [SEP]
ROUTINE (COMPILER) [BS] ≡ LET [PI] ≡ [RESOLVED-P][SEP]
ROUTINE (COMPILER) [BS] ≡ -> [LABEL][IU][PI]  ≡ [RESOLVED-P][SEP]
ROUTINE (COMPILER) [BS] ≡ ->[LABEL][SEP]
ROUTINE (COMPILER) [BS] ≡ SHIFT [AB] UP [ABN] [SEP]
ROUTINE (COMPILER) [BS] ≡ SHIFT [AB] DOWN [ABN] [SEP]
ROUTINE(COMPILER)[BS]≡[FD][COMMA][WORD/1][COMMA][WORD/2][COMMA][WORD/3][SEP]
ROUTINE (COMPILER) [BS] ≡ ([ADDR]) = [WORD/1][OPERATOR][WORD/2][SEP]
ROUTINE(COMPILER) [BS] ≡ END[SEP]
ROUTINE (COMPILER) [BS] ≡ [AB/1] = [WORD][SEP]
ROUTINE(COMPILER) [BS] ≡ ([ADDR]) = [WORD] [SEP]
ROUTINE (COMPILER) [BS] ≡
PLANT[FD][COMMA][ABN/1][COMMA][ABN/2][COMMA][WORD]IN[B][SEP]
```

```
ROUTINE (COMPILER)[BS] ≡ ->[LABEL][IU][WORD/1][COMPARATOR][WORD/2][SEP]
ROUTINE (COMPILER)[BS] ≡ [AB] = [WORD/1][OPERATOR][WORD/2][SEP]
ROUTINE (COMPILER)[AS] ≡ ADD [WORD] TO [LIST OR NEST][AB][SEP]
ROUTINE (COMPILER) [BS] ≡ [AB] = CATEGORY OF [PI] [SEP]
ROUTINE (COMPILER) [BS] ≡ [AB] = ADDRESS OF [PI] [SEP]
ROUTINE (COMPILER) [BS] ≡ LET [PI] ≡ [RESOLVED-P][SEP]
ROUTINE (COMPILER) [BS] ≡ -> [LABEL][IU][PI] ≡ [RESOLVED-P][SEP]
ROUTINE (COMPILER) [BS] ≡ SHIFT [AB] UP [ABN] [SEP]
ROUTINE (COMPILER) [BS] ≡ SHIFT [AB] DOWN [ABN] [SEP]
ROUTINE (COMPILER)[BS] ≡ ->[LABEL][SEP]
ROUTINE(COMPILER)[BS]≡[FD][COMMA][WORD/1][COMMA][WORD/2][COMMA][WORD/3][SEP]
ROUTINE (COMPILER) [BS] ≡ ([ADDR]) = [WORD/1][OPERATOR][WORD/2][SEP]
ROUTINE(COMPILER) [BS] ≡ END[SEP]
ROUTINE (COMPILER)[BS] ≡ PRINT SYMBOL[ABN][SEP]
ROUTINE (COMPILER) [BS] ≡ [AB] = NUMBER OF [PI][SEP]
ROUTINE(COMPILER) [BS] ≡ [AB] = CLASS OF [PI][SEP]
ROUTINE (COMPILER)[BS]≡INDEX [ABN] = [AB][SEP]
ROUTINE (COMPILER) [BS]≡[AB]= INDEX [ABN][SEP]
ROUTINE (COMPILER) [BS] ≡ SPACE [SEP]
ROUTINE (COMPILER)[BS] ≡ NEWLINE [SEP]
ROUTINE (COMPILER)[BS] ≡ CALL R [ABN] [SEP]
ROUTINE (COMPILER) [BS] ≡ PRINT [ABN][SEP]
ROUTINE (COMPILER)[BS]≡ [FD][COMMA][WORD][COMMA] 0 [COMMA] L [LABEL]
ROUTINE (COMPILER)[BS]≡[WORD/1]/[WORD/2][SEP]
ROUTINE (COMPILER) [BS] ≡ PRINT [ABN] IN OCTAL [SEP]
ROUTINE [MP] ≡ FORMAT CLASS
ROUTINE [MP] ≡    BUILT-IN PHRASE
BUILT-IN PHRASE   [ALL SYMBOLS EXCEPT RT BRACKET]
ROUTINE     R 180
FORMAT [SS] =     POPI [WORD] TO [WORD][EOL]
ROUTINE [SS] ≡ POPI [WORD/1] TO [WORD/2][EOL]

END OF MESSAGE

PHRASE [NOTE']          =     [COMMA]|[NOTE],NIL
PHRASE [TABLE LABEL]    = [N]:[TABLE LABEL],NIL
FORMAT [AS] = PLANT [AB]TH ORDER OF TABLE [TABLE] IN [B][SEP]
FORMAT [AS] = PLANT TABLE [TABLE] IN [B][SEP]
FORMAT [AS] = |[NOTE][SEP]
BUILT-IN PHRASE   [NOTE]
ROUTINE (COMPILER) [AS] ≡     |[NOTE][SEP]
PHRASE[TABLE]  =  [TABLE ENTRY][REST OF TABLE]
PHRASE [TABLE ENTRY] = [68][TABLE
LABEL][FD][COMMA][αβN][COMMA][αβN][COMMA][WORD][NOTE']
PHRASE[REST OF TABLE] = [TABLE ENTRY][REST OF TABLE],NIL
ROUTINE(COMPILER)[AS] ≡ PLANT[αβ]TH ORDER OF TABLE [TABLE]IN[β][SEP]
ROUTINE(COMPILER) [AS] ≡ PLANT TABLE [TABLE] IN [β][SEP]
ROUTINE [AS] ≡ [AB/1] = LIST [AB/2]([ABN/3],?)[SEP]
ROUTINE [AS] ≡ MONITOR ([ALL SYMBOLS EXCEPT RT BRACKET])[SEP]
ROUTINE [AS] ≡ LIST[AB] = NEXT RECONSTRUCTED LINE[SEP]
ROUTINE [AS] ≡    [AB] = [LIST OR NEST][WORD][SEP]
ROUTINE [AS] ≡    [AB] = [LIST OR NEST]([WORD/1][, WORD*])[SEP]
ROUTINE [AS] ≡    [AB/1] = VALUE OF LIST [AB/2] IN DICT [AB/3][SEP]
ROUTINE [AS] ≡    DELETE CONVENTIONAL LIST [AB][SEP]
ROUTINE [AS] ≡    DELETE LIST [AB/1] FROM DICT [AB/2][SEP]
ROUTINE [AS] ≡    WITHDRAW [AB/1] FROM NEST [AB/2][SEP]
ROUTINE [AS] ≡    LIST [AB/1] = LIST [AB/2] + LIST [AB/3][SEP]
ROUTINE [AS] ≡    LIST [AB/1] = ENTRY WITH VALUE [AB/2] IN DICT [AB/3][SEP]
ROUTINE [AS] ≡ ADD ([WORD][, WORD*]) TO [LIST OR NEST] [AB][SEP]
ROUTINE [AS] ≡ [AB] = CONVENTIONAL LIST OF [ABN] WORDS [SEP]
```

```
ROUTINE [AS] ≡    ASSIGN VALUE [ABN] TO [PI][SEP]
ROUTINE [AS] ≡ PRINT LIST [ABN] [SEP]
ROUTINE [AS] ≡ ANALYSE LIST [AB] W.R.T. [PI] [SEP]
ROUTINE [AS] ≡ ADD LIST [AB/1] [COMMA] [WORD] TO DICT [AB/2] [SEP]
ROUTINE [AS] ≡ DELETE [LIST OR NEST] [AB] [SEP]
ROUTINE (COMPILER)[AS] ≡ [AB/1] = LIST [AB/2]([ABN][COMMA]?)[SEP]
ROUTINE (COMPILER)[AS] ≡ DELETE [LIST OR NEST][AB][SEP]
ROUTINE (COMPILER)[AS] ≡ LIST [AB] = NEXT RECONSTRUCTED LINE [SEP]
ROUTINE (COMPILER) [AS] ≡ [AB] = [LIST OR NEST][WORD][SEP]
ROUTINE (COMPILER)[AS] ≡ ADD ([WORD][,WORD*]) TO [LIST OR NEST][AB][SEP]
ROUTINE (COMPILER) [AS] ≡ [AB] = [LIST OR NEST]([WORD][,WORD*])[SEP]
ROUTINE (COMPILER)[AS] ≡ WITHDRAW [AB/1] FROM NEST [AB/2][SEP]
ROUTINE (COMPILER)[AS] ≡ LIST [AB/1] = LIST [AB/2] + LIST [AB/3][SEP]
FORMAT [AS] = [αβ] = UPPER LIMIT OF [αβ] IN DICT [αβ][SEP]
ROUTINE [AS] ≡    [αβ/1] = UPPER LIMIT OF [αβ/2] IN DICT [αβ/3][SEP]
FORMAT [AS] =     [αβ] = LIST FOR [αβ] IN CLASS [αβN][SEP]
ROUTINE [AS] ≡    [αβ] = LIST FOR [αβ/1] IN CLASS [αβN][SEP]
ROUTINE [AS] ≡ [αβ] = LIST [PI][SEP]
ROUTINE (COMPILER) [AS] ≡ MONITOR ([ALL SYMBOLS EXCEPT RT BRACKET]) [SEP]
ITEM 274
REPLACE ITEM 150
FORMAT [AS] = PRINT B-LINES [SEP]
ROUTINE (COMPILER) [AS] ≡ PRINT B-LINES [SEP]
ROUTINE SMALL R 283
DELETE ITEM 257
ROUTINE R 257
DELETE ITEM 258
ROUTINE R 258

END OF MESSAGE

ROUTINE [MP] ≡ DEFINE COMPILER
ROUTINE R 226
ROUTINE R 237
ROUTINE [AS] ≡ CONVERT [PI] TO [AB] [SEP]
FORMAT [AS] =   CALL BUILT-IN PHRASE [PI][SEP]
FORMAT [AS] =   PRESERVE ANALYSIS B-LINES [SEP]
FORMAT [AS] =   RESTORE ANALYSIS B-LINES [SEP]
ROUTINE(COMPILER)[AS] ≡ CALL BUILT-IN PHRASE [PI][SEP]
ROUTINE(COMPILER)[AS] ≡ PRESERVE ANALYSIS B-LINES [SEP]
ROUTINE(COMPILER)[AS] ≡ RESTORE ANALYSIS B-LINES [SEP]
FIXED ITEM 0
FIXED ITEM 281
FIXED ITEM 135
FIXED ITEM 279
I280
REPLACE ITEM142

END OF MESSAGE


DEFINE COMPILER CC1


COMPILER COMPILER (CC1)
```

APPENDIX 4

The Compiler Compiler Index

COMPILER COMPILER
INDEX

	Key to Types	tag in index
M.	Miscellaneous item	00
R	Large routine	00
r	Small routine	01
P	Phrase	10
PR	Phrase routine (built-in)	11
O	Index register is left clear - type is determined by the way it is used	00
S	Special purpose (0-16 only)	-
A	Administrative or Interpretive Routine	01

Type	Index		
SP	0	0101, 127 0	Compiler entry point (Item 0 has
SP	1	150 *2	a special significance - see item 278)
SP	2	origin of record score (B88)	
SP	3	head of index chain (B87)	
SP	4	origin of stack (B90) (set by R161 for use by R257)	
SP	5	Current set of output (=0 I.S.; =64 O.S)	
SP	6	end of xed part of store	
SP	7	1102, 70,74	
SP	8	Used by instructions involving B121, B122	
SP	9		
SP	10		
SP	11		
SP	12		
SP	13	Address of start of chain store (reserved for)	
SP	14	Length of chain store (used by R161. set by SET[N]BLOCKS)	
SP	15	Tape address of compiler (c.f.1147 extracode)	
SP	15	List program marker	
	17	available to user	
.	.	.	
.	.	.	
.	.	.	
.	.	.	
0	128	available to user	

TYPE	Paper Tape	Index		
M	C	129	o	Conventional list of 20 format classes including [AS], [SS] [BS], [MP]
P	B,C	130	†	[MP] = PHRASE, ITEM, END OF MESSAGE, FORMAT, FORMAT CLASS, DELETE ITEM, REPLACE ITEM
P	C	131	o	[BS] dictionary (M x dict) |In chain
P	C	132	o	[AS] dictionary (M x dict) |store during
P	C	133		[SS] dictionary (M x dict) |input of primary
M	C	134	o	CID dictionary (no M word) material
R	H	135	*	Source material analysis routine
M	-	136		Max. no. of lines allowed in [SS]
R	B	137		Dummy Routine
M	-	138		No. of faults detected by Master Routine
M	D	139	o	skeleton of A* and A? A* ≡ A* = A A* COMMA A EOS A? ≡ A? = A COMMA NIL EOS
M	-	140		Max. no. of faults to be allowed by Mater Routine
r	D	141		look up or enter in a double-entry list
R	C	142	‡	pre-editing routine
P	D	143	o	(CR),NIL
P	E	144	o	[label] = [separator*?] EOS , [separator*?][primary label]
R	C	145		DELETE ITEM routine
O	G	146	o	[primary label] = [N]),NIL
O	G	147		[instruction] = [BS] , [182][AS], [182][SS]
PR	E	148	o	[separator*?] = COMMA , EOL (record contracted ant)
PR	D	149		[N] built in
R	B,H	150	*	Initial entry routine
P	D	151	o	[body of a phrase defn.] : [143][ ] = [phrase*]
PR	D	152	o	[ ] built in = any phrase identifier AR =&I
O	G	153		[AB] or [β]
O	G	154		[ABN] or [βN]
R	C	155	†	delete an item in the store
P	D	156	o	[phrase*] = [phrase][phrase*] , [phrase] EOS
P	D	157	o	[phrase] = BUT NOT , [or ES*]
PR	D	158	o	[or ES*] built in (Any sequence of identifiers or basic symbols) AR = & B I I I

Type	Tape	Index		
P	D	159	o	[serial number] = COMMA [N], COMMA[or ES*], NIL
PR	D	160	*	[N] (as 149 but not in CID), used by 146
R	B,C	161	*	Set Chain & Stack (subroutine of 150)
O	-	162		User's Entry
P	D	163	o	[body of format defn.] = [pi] = [pi or ES*][serial number]_EOS_
R	C	164	o	REPLACE Item Routine
M	-	165		Temporary index register used by 164, 237
PR	D	166		[a] and [A] built-in : effectively 1, 2,,,,, 4194303 or A1,A2,,,,,
PR	D	167		[b] and [B] built-in : effectively β1,β2,,,,,β4194303 or B1,B2,,,,,
P	-	168		spare
M	C	169	o	list of dictionaries and lists to be packed
A	F	170	*	interpret -> B82
P	D	171	o	[general phrase identifier] = [[phrase identifier][phrase label] [phrase index]]
PR	D	172	o	[phrase identifier] built-in AR = X BnIIII or X N
PR	E	173		[OW]
P	D	174		[Phrase label] = / [N], NIL
R	C	175		Second entry to 'delete item' (155) routine or list of routines to be deleted by END OF PRIMARY MATERIAL
P	D	176	o	[phrase index] = ([ABN]), NIL
r	D	177	o	print new format or phrase
O	G	178		[0-3]
P	E	179		[SEP] = COMMA, EOL
O	G	180	‡	diagnostic subroutine of Master Routine
PR	E	181	o	[PI] denotes identifier / label?/index?
O	G	182	o	(CR)[182] = *, NIL
r	D	183		Print label
PR	E	184	o	[LABEL]
P	E	185	o	[RESOLVED-P] = [set p] [reset p] 1, [reset p] -1
O	G	186	o	[GENERATED-P] [186] = [152]

Type	Tape	Index		
A	F	187	╪	[AB] = [WORD]
A	F	188	╪	[AB] = [WORD][OPERATOR][WORD]
A	F	189	╪	([ADDR]) = [WORD]
A	F	190	╪	([ADDR]) = [WORD][OPERATOR][WORD]
A	F	191		[JUMP][LABEL][IU][WORD][COMPARATOR][WORD]
A	F	192		[JUMP][LABEL]
A	F	193		LET [PI] = [GENERATED-P]
A	F	194		[JUMP][LABEL][IU][PI][EQV][RESOLVED-P]
A	F	195		LET [PI][EQV][RESOLVED-P]
A	F	196		[JUMP][LABEL][IU][PI] = [PI]
A	F	197		[AB] = CATEGORY OF [PI]
A	F	198		[AB] = NUMBER OF [PI]
A	F	199		[FD][COMMA][WORD][COMMA][WORD][COMMA][WORD]
A	F	200		PLANT [FD][COMMA][WORD][COMMA][WORD][COMMA][WORD]IN [AB]
A	F	201		[PI] = [AB]
A	F	202		[AB] = ADDRESS OF [PI]
A	F	203		[AB] = CLASS OF [PI]
r	D	204		add word to list
r	D	205		add word to nest
r	D	206		withdraw word from nest
r	D	207		delete chain
A	F	208		CALL R [ABN]
r	D	209		add list to list
r	D	210	o	copy linear list to chain
A	F	211		CALL R [PI]
A	F	212		[FD][COMMA][WORD]0[COMMA]L[LABEL]
R	D	213	o	Read next section
R	B,C	214	¥	Master routine
R	B,C	215		Analysis routine
M	C	216		24-bit multiplication and division routine
r	E	217	o	Convert absolute &'s to relative &'s

Type	Tape	Index		
R	D	218	o	PHRASE routine
r	C	219		print B82 in Octal
R	D	220	o	FORMAT routine
R	E	221	o	ROUTINE routine
R	B	222	o	Dual routine
r	E	223	o	'Is it parameter-free?' routine
R	D	224		General Dict. Routine
R	D	225	o	Merge new entry into dictionary (first or last)
O	H	226	†	END OF PRIMARY MATERIAL routine
R	C,C	227		END OF MESSAGE routine
r	C	228	o	Transfer dict. to record store
P	D	229	o	COMMA
R	D	230	o	[identifier] conversion routine
r	C	231	o	transfer dict. to chain store
A	F,H	232	*	Transplant routine (B74)
A	F	233		look-up
r	F	234		look-up ([PI]) L.S.E. routine
A	F	235		Compute value of a word
A	F	236		Compute value of an address
O	H	237		Body of end of primary material routine
R	B,C	238	¥	Line reconstruction routine
M	B,H	239	*	DOWN sequence(B76)
M	B	240	*	END sequence
R		241		List B61 = entry with value B63 in dict. B62
R	D	242	o	Auxiliary Phrase routine
R	C	243		General Packing Routine for dicts. etc.
PR	E	244	o	Set p'
R	C	245		General unpacking routine
M	B,C	246	+++	Standard flexwriter tab-settings
M	B	247	+++	Line image for R238

Type	Tape	Index		
r	c	248		Decimal printing routine
C	G	249	o	[compile phrase] = (COMPILER), NIL
PR	E	250	o	reset p
PR	E	251		[FD]
r	B	252	*	Split chain into 2 sub-chains
R	E	253	o	Compile body of a routine
O	G	254	o	[routine heading] = SMALL R [N], R [N],[compile phrase] [PI][EQV][GENERATED-P]
PR		255		Spare for phrase routine
M	C	256	o	Double entry list for serial no of routine / serial no of corresponding compiled version
R	C,H	257	*	Catastrophic Fault Routine
R	C,H	258	*	Non-catastrophic fault routine
r	D	259	o	add nil branch to dictionary
R	D	260	o	Entry to phrase routine used by auxiliary phrase routine
R	B,C	261	¥‡	Convert metasyntactical symbols
A	F	262		END (i.e. the [BI] format)
A	F	263		[AB] = INDEX [ABN]
A	F	264		INDEX [ABN] = [AB]
r	D	265	o	delete superfluous EOL's
R	B	266	o	Item routine
A	F	267		SHIFT [AB] UP [ABN]
A	F	268		SHIFT [AB] DOWN [ABN]
A	F	269		SPACE
A	F	270		NEWLINE
A	F	271		PRINT [ABN]
O	G	272	o	FORMAT CLASS routine
A	F	273		PRINT SYMBOL [ABN]
O	H	274	*	Sequence used by monitor
O	H	275		DEFINE COMPILER
O	-	276	+-+	Body of Define Compiler, Define Master Compiler, Define Special Compiler
A	F	277		PRINT [ABN] IN OCTAL
R	E	278	o	FIXED ITEM routine
O	H	279	*	Fixed 'DOWN' sequence (B76)
O	H	280	*	Fixed 'TRANSPLANT' sequence (B74)

```
Type Tape    Index

O    H       281 *     SLIDE COMPILER UP A BLOCK (FIXED ITEM)
M            282 *     Compiler title record (set in DEFINE COMPILER: printed in R150)
             283       PRINT B-LINES (as preserved on stack)
             284
             285
             286
             287
             288
             289
             290
             291
O    G       292       BUILT-IN PHRASE routine
             293
             294
             295
R            296 *     Error routine[call for a deleted routine]
r            297 +++    Print character (subroutine of R238)
O            298 *     TRACE for END (Fixed itemO
O            299       TRACE for DOWN
```

Type	Index		
P	300	o	[EQV]
P	301		[JUMP]
P	302	o	[OPERATOR]
P	303	o	[COMPARATOR]
P	304		[ADDR]
P	305		[WORD]
P	306		[-?]
P	307		[-]
P	308	o	[IU]
P	309	o	[,WORD]
P	310	o	[LIST OR NEST]
O	311		[WORD]/[WORD] (only compiling version exists)
R	312		[AB]=[LIST OR NEST][WORD]
P	313	o	[,WORD*]
R	314		[AB]=[LIST OR NEST][WORD]
R	315		[AB]=LIST[PI]
R	316		[AB]=LIST[AB]([ABN*?])
R	317		[AB]=VALUE OF LIST[AB]IN DICT[AB]
R	318		[AB]=CONVENTIONAL LIST OF [ABN] WORDS
R	319		DELETE CONVENTIONAL LIST [AB]
R	320		DELETE [LIST OR NEST][AB]
R	321		DELETE LIST[AB]FROM DICT[AB]
R	322		ADD([WORD][,WORD*])TO[LIST OR NEST][AB]
R	323		ADD[WORD]TO[LIST OR NEST][AB]
R	324		ADD LIST [AB],[WORD] TO DICT[AB]
R	325		ASSIGN VALUE[ABN]TO[PI]
R	326		ANALYSE LIST[AB] W.R.T.[PI]
R	327		WITHDRAW[AB]FROM NEST[AB]
R	328		LIST[ABLIST[AB]+LIST[AB]
	329		LIST[AB]=ENTRY WITH VALUE[AB]IN DICT[AB]
	330		LIST[AB]=NEXT LINE FROM INPUT[N]
R	331		LIST[AB]=NEXT RECONSTRUCTED LINE

Type	Index		
R	332		CONVERT[PI]TO[AB]
R	333		CALL[PI]COMPILER[GENERATED-P]
R	334	o	[ALL SYMBOLS EXCEPT RT BRACKET]
R	335		MONITOR ([ALL SYMBOLS EXCEPT RT BRACKET])
R	336		PRINT LIST[ABN]
CV	337		
CV	338		
CV	339		
R	340		DEFINE MASTER COMPILER
CV	341		
CV	342		
CV	343		
CV	344		
CV	345		
CV	346		
CV	347		
CV	348		
CV	349		
CV	350		
CV	351		
CV	352		
CV	353		
CV	354		
CV	355		
CV	356		
CV	357		
CV	358		
CV	359		
CV	360		
CV	361		
CV	362		
CV	363		

Type	Index		
CV	364		
R	365	o	POPI[WORD]TO[WORD]
P	366	o	[NOTE']
P	367	o	[NOTE]
P	368	o	[TABLE LABEL]
R	369		[TABLE]
R	370		PLANT[AB]TH ORDER OF TABLE[TABLE]IN[B]
R	371		PLANT TABLE[TABLE]IN[B]
R	372		|[NOTE] (only compiling version exists)
CV	373		
P	374		[TABLE ENTRY]
P	375		[REST OF TABLE]
CV	376		
CV	377		
CV	378		
CV	379		
CV	380		
CV	381		
CV	382		
CV	383		
CV	384		
CV	385		
R	386		[AB]=UPPER LIMIT OF[AB]IN DICT[AB]
R	387		[AB]=LIST FOR [AB]IN CLASS[ABN]
CV	388		
R	389		PRINT B-LINES (only compiling version exists)
CV	390		
R	391		CALL BUILT-IN PHRASE[PI] (only C.V.exists)
R	392		PRESERVE ANALYSIS B-LINES (only C.V.exists)
R	393		RESTORE ANALYSIS B-LINES (only C.V.exists)
CV	394		
CV	395		

Type	Index		
CV	396		
CV	397		
R	398	†	DEFINE SPECIAL COMPILER
R	399		LIST PROGRAM
R	400		DO NOT LIST PROGRAM
R	401		NOTES:
P	402		[LNOTE']
PR	403		[LNOTE]
R	404		SET[N]BLOCKS
R	405		FRIG
CV	406		
P	407		FORMAT CLASS [DEBUG]
P	408		[DEBUG OPTS]
R	409		FORMAT [AS]=[DEBUG]
R	410		PRINT ITEM[WORD]
R	411		RPRINT[WORD]
R	412		[AB]=END OF ITEM[WORD]
R	413		SWITCH TRACE ON
R	414		SWITCH TRACE OFF
R	415		POPO[WORD]TO[WORD]
CV	416		
	417		
	418		
	419		
	420		
	421		
	422		
	423		
	424		
	425		
	426		

Note that the Type CV indicates a Compiling Version. Such routines are given a new serial number each time they are redefined and the old number is returned to the list of available indexes. The correspondence between a CV and its associated Format number is given by Item 256.

FOOTNOTE KEY

*	Must not be deleted
†	Can be deleted only by calling R155 from R276
‡	Can be deleted only by rewriting the master routine R214
±	These four routine are really the same routine. One may not be deleted unless all 4 are deleted and then only by deleteing one of them and then clearing the indices of the other 3.
¥	Can be rewritten to provide shorter versions in a finalised compiler
+-+	It is possible to write a self deleting verion of R276
+++	Can be deleted by rewriting R238
o	Deleted by the standard END OF PRIMARY MATERIAL
	Receives special attention in END OF PRIMARY MATERIAL