2. THE COMPILER-COMPILER BOOTSTRAP

2.1. Introduction

To initialize the C-C so that it is capable of self-extension and therefore able to build itself up to the required level we need a set of routines, tables and dictionaries which must be provided in a language independent of the compiler. This set is as follows:-

a.	INDEX	
b.	INITIALIZATION ROUTINES	150 and 161
c.	INPUT ROUTINES	238 and 261
d.	MASTER ROUTINE	214
e.	MASTER PHRASE DICTIONARY	[MP] 130
f.	DOWN and UP	239 and 240
g.	ANALYSIS ROUTINE	215 with its subroutines 222 and 252
h.	ITEM ASSEMBLY ROUTINE	266
i.	VARIOUS MACHINE DEPENDENT ROUTINES	

These routines need not be in their final form but must be capable of recognizing and generating ITEM routines. The final forms of these routines can then be defined as ITEM routines replacing these temporary versions. However, DOWN and UP will be inserted in their final forms as most of the mechanism is required; also the INDEX and routines 238 and 266 which are machine dependent and will not be changed.

2.2. INDEX

The INDEX is a set of fixed locations which we will denote by X_0, X_1, etc., up to a figure larger than X300 and less than X1024. These provide locations for storing fixed global parameters of the system in a similar manner to the B variables. Their main purpose, however, is to contain the entry points to all routines and dictionaries. All routine and dictionary accesses are made via the address in their index position so that movement of a routine only requires changing the contents of the index position. Each routine has a number associated with it which corresponds to its index position. Besides the address of the routines each index entry contains a two-bit flag field F which is used to differentiate between different types of entries:-

F_0	Routines
F_1	Small Routines with simplified entry and exit
F_2	Dictionaries and Phrase definitions
F_3	Built-in Phrase Definitions.

As the Item Assembler is capable of defining all 4 types of entries in the record store but without the mechanism for differentiating between them, the Flag bits for entries being provided by the Item Assembler must be preloaded into the Index. Also, of course, the entry points for the routines in the boot-strap must be inserted in the Index.

The complete list of entries necessary is as follows:-

INITIALIZATION PARAMETERS

X_0	On Atlas this contains an instruction which causes entry to the routine X150. This is due to Atlas requiring entry at the start of a block initially. If entry is made direct to X150 then this entry can be left empty.
X_2	Current origin of Record Store., i.e., pointer to next available location in the record store area.
X_3	= 300, the head of the Index Chain of unused Index Positions.
X_7	= Pointer to set of instructions loaded before an analysis record to cause the analysis record to be obeyed.
X_9	= 1024, the length of the Index.
X_{10}	= Numerical value of the EOL character. This will be used as a word to test against in the ITEM routines.
X_{11}	= Numerical value of the character zero. It will be assumed that the characters for the remaining digits have numerical values immediately above this. Therefore, for example, the value of

	character '9' in X_{11} + 9.
X_{12}	= Numerical value of the $ character (or some other character) which is to be used to signify that the remainder of an input line, including this character, is to be treated as a comment.
X_{13}	= This location is used to store the address of a replacement ITEM while the new ITEM is being compiled.
X_{14}	= The entry pointer B71 is stored here on entering a Small routine. It is all that needs to be saved. Note that in general, a Small routine cannot call other routines.
X_{15}	= 0 for obtaining a listing of the input.
X_{16}	= Pointer to table of Operation Codes on the G-21. It has been used as a pointer to allow the G-21 PLANT routine to access the parts of a G-21 opcode that are required.
X_{17}	= Stack Origin.
X_{18}	= Maximum Chain Address.
X_{136}	= 3, maximum number of lines allowed in a Source Statement.
X_{140}	= 10, maximum number of faults to be allowed by [MR].

The routines defined in the bootstrap must also have their entry points loaded in the INDEX. These are 130(F2), 150, 161(F1), 214, 215, 216(F1), 222, 238, 239, 240, 241(F1), 245, 248(F1), 252(F1), 261(F1), 266, 275, 283(F1). The ITEM routine also has a subsidiary entry point at position 168 which is used to enter the routine for finding an integer arriving from the input.

In the original Atlas implementation storage was left following the INDEX so that a fast non-relocatable form of the Source Statement Analysis Routine and DOWN sequence could be inserted. This is not essential, but if implemented in this way, then storage should be reserved at this point. As it is not required until the complete Compiler has been produced, we will leave discussion of this until later.

Flag bits for the following positions must be set:-

F_1 X170, 177, 183, 187 to 212, 216, 217, 219, 223, 228, 231 to 236, 248, 259, 262 to 265, 267 to 271, 273, 277

F_2 X131, 132, 133, 142, 151, 156, 157, 159, 163, 168, 171, 174, 176, 179, 185, 229

F_3 X148, 149, 152, 158, 160, 166, 167, 172, 173, 181, 184, 244, 250, 255

Finally the free Index positions are chained together starting at X300. Thus X300 contains 301, X301 contains 302, and so on. It is conventional to leave the last 20 or 30 Index Positions as a set of Small Routine positions for the user and these would not be chained. On Atlas these were positions 1000 to 1024. The last chained Index was therefore X998 which contained 999 and X999 was set to zero to indicate exhaustion of chain. The number of Index Positions depends only on the amount of storage available.

2.3. Initialization Routines

The initialization routine 150 together with its subroutine 161 define the storage layout of the compiler and initialize the Stack and Chain Store.

At this stage all that is required in Routine 150 is:-

a. Set B76 = Entry point of the DOWN routine.
b. Call R161
c. Enter MASTER ROUTINE

In general the user would not be expected to alter R150 but he could alter R161. (For example he might require a longer chain).

Routine 161 defines the positions of the stack, chain store and record store. In a machine with a conventional storage layout it would be reasonable to have:-

a. At the lowest address position would be space allocated to the 127 B variables follwed by the INDEX table. Following this room would be left for the source statement analysis routine and then the bootstrap routines defined above. The record store will then be from this point up the store.

b. Near the top end of the store will be allocated an area to be used both by the stack and Chain Store. The CC is written so that increasing core addresses correspond to higher locations in the stack. The stack is therefore designed to eat into the chain store which is chained up so that the highest address on the chain will be used first (a description of chains is given in [6]). This does not give us any protection against the stack and chain running into each other but is sufficient until more complex routines can be added for storage allocation and protection.

If we assume the region extends from location N to M then we require:-

B72 = N	Base of working area
B90 = N+1	Stack Pointer
B89 = M-1	Chain Pointer

The Chain should be linked as follows:-

Address	Contents
N+3	
N+4	N+1
N+5	
N+6	N+3
...	...
...	...
M-4	M-7
M-3	
M-2	M-5
M-1	
M	M-3

In addition Routine 161 sets the following:-

X4 = B90	
X138 = 0	Number of faults detected by Master Routine
X140 = 10	Maximum number of faults allowed by Master Routine
B74 = X232	The entry address of the transplant routine.
B86 = 0	The line number
B87 = 300	The first free Index position
B88 = X2	First free address in record store

2.4. Input Routines

These routines will depend largely on the particular machine available. The Compiler Compiler restricts the number of possible input characters to 127 and these are assigned the numerical values 1 to 127. The correspondence between particular characters and numerical values is immaterial and can be chosen to suit the particular peripheral equipment or internal representation of a machine.

Routine 238 is designed to read the next section of input into a circular chain list (as described in reference [6]). Each chain position should contain a number between 1 and 127 corresponding to the character found on the input. The newline character should be assigned a value. It is unfortunate that some characters may be referred to specifically by their numerical values in the Item routines. The most likely characters involved in this way will be the Newline, Space, Comma and [. It is hoped to eventually remove these by having indirect references to these via the INDEX. The amount of input in a section is up to the user but the Item Routines assume one instruction per section so that having a section equivalent to a line is a reasonable choice initially. R238 exits with the next section in a circular chain list pointed at by B61. In addition the line number B86 is updated.

Routine 261 is used to convert some characters or character strings to unique special meta-syntactic characters required for the analysis of a line with respect to the Master Phrase dictionary. In particular it removes any redundant characters. Space and erase are non-significant for example when analysis takes place with respect to the [MP] dictionary. Also, it sets the numerical value for [and , to special values which will be needed for checking against. In the original version the position 4 for EOL (End of Line), 5 for [and 105(8) for COMMA were chosen. On the G21 104, 106 and 105 octal were chosen, respectively. At the moment I am unclear how necessary this change is. To allow parts of a line to act as comment all characters after and including the one whose numerical value is stored in X_{12} is ignored. ($ will be used initially

The output form R261 is, in general, a reduced and modified form of the original string. The Routine expects B69 to point to the input chain list and on exit will have built up the reduced line and have B68 pointing at it. The original line is unchanged and may be required if, for example, the statement is not to be analyzed with respect to the Master Phrase Dictionary. It may be worth introducing at this stage some comment convention for ignoring parts of the input section.

2.5. Master Routine and [MP] Dictionary

Initially all that is required of the Master Routine is to be able to recognize the one entry in the [MP] dictionary which is ITEM. However, before the [MP] dictionary can be replaced by its final form it will be necessary to define a routine for replacing ITEM routines. Therefore, a second phrase REPLACE ITEM is also required to be recognized. Using the notation of the instructions in the Item Routine and the form of dictionaries defined in [5] we have for the Master Phrase Dictionary 130:-

```
H = 56 + J2          M - word
H = 19 + F1          & - word
H = 7 + F1           & - word
C = I
C = T
C = E
C = M
H = 266              Index Position of ITEM Assembler
C = R
C = E
```

C = P	
C = L	
C = A	
C = C	
C = E	
C = I	
C = T	
C = E	
C = M	
H = 164	Index Position of Replace ITEM routine
H = 0	
H = 0	

We are now in a position to define the action of the Master Routine 214. Brackets will be used in the Compiler-Compiler convention. For example (B61 + 1) means the contents of the address B61 + 1. We have, therefore:-

1. Enter Input Routine 238
2. If line empty go to <u>1</u> [B61 = (B61+1)]
3. B85=B69=B61 and enter Conversion Routine 261
4. B61 = B68 = condensed form of input line
5. B62 = 130, [MP] dictionary position
6. B63 = 0, denotes primary entry to analysis routine
7. Enter Analysis Routine 215
8. If B64 ≠ 1 then no recognition and we have an error
9. B79 = B61, unrecognized part of line
10. Return original line pointed at by B85 and recognized part of line pointed at by B62 back to Main Chain
11. B77 = B63, analysis record of 'ITEM'
12. B60 = B77 - 5 (This may not be necessary)
13. Enter routine whose position in Index is (B77)
14. If B79 = 0 then no input remains to be analyzed and go to 1
15. If B79 ≠ 0 set B61 = B79 and go to 5.

2.6. DOWN and UP Routines

These two routines will not be changed later on as most of the mechanism is necessary immediately, and so it is convenient to store them in their final form initially.

There are basically two different entries to the DOWN routine. The purpose of the DOWN routine in the normal entry is to update the stack before entering the new routine. Back and forward links are inserted in the stack and work space for the new routine is allocated. Sufficient information is stored so that the UP routine can restore the state of the stack to its initial form on exit from the routine.

The subsidiary entry is the TRANSPLANT entry described in [5]. The main nodes of the analysis record are moved into the stack and the associated routine is entered. This will be described in more detail later.

The normal entry is as follows:-

1. B60 = B90, temporary storage for stack pointer B90
2. B77 = -1, switch to denote normal entry
3. B92 = (B70-i) = Routine Number

   The actual macro for entering the DOWN sequence must contain the routine number to be executed. B70 must be the return pointer and B70-i here is used to denote the storage position of the Routine Number. This is machine dependent and the user can make his own choice.
4. B91 = contents of Index position B92, the routine entry point
5. If F1 bit of B91 set then small routine. Set B60 = B91 and enter routine in this case. Store old B71 in X_{14}. Set B71 = B91.
6. If not small routine then:-

(B72) = B90	Forward link
(B90) = B70 - B71	Relative position of return in old routine
(B90+1) = B75	Old routine number

(B90+2) = B60	Backward Pointer
(B90+3) = B72	Origin of workspace in old routine
B71 = B91	Location of current routine
B75 = B92	New routine number
B72 = B90+4	New forward link position and origin of workspace
B73 = B71 + (B71+1)	Location of label directory for routine
B90 = (B71) + B72+1	New stack pointer position having moved over workspace

7. B60 = B91 and enter routine.

If the DOWN routine is entered for TRANSPLANT then entry is made at 4. In this case B77 will contain the position of the analysis record (≥ 0) and B92 set to Index position of analysis record. In this case between 6. and 7. is inserted:-

```
6.1  (B72+1) = B77
     (B72+2) = (B77+1)
     (B72+3) = (B77+2)
     etc.
```

until all &'s of top level of analysis record have been copied into stack; that is until a word without F1 set is encountered. The rest of the workspace area up to B90 is then initialized to zero. This will be discussed in more detail later.

The UP routine returns stack to the position before entry to the previously called routine which we are now wxiting from. The stack must be adjusted as follows:

B70 = (B72-4)	Recover relative position of return
B75 = (B72-3)	Routine number
B90 = (B72-2)	Stack Pointer
B72 = (B72-1)	Pointer to storage area

B71 = Contents of Index position B75 = entry position of the old routine.
B73 = (B71+1) + B71 = Location of label directory
Transfer to B71 + B70.

2.7. Analysis R~~ecord~~outine

This is only required to recognize ITEM and REPLACE ITEM so far so that it can have the following simplified form:-

```
1.  B43 = B62
2.  B62 = B61 ∧ A                 (A = address mask)
3.  B63 = B88
4.  B68 = B90
5.  B67 = (Contents of Index B43) ∧ A
    B65 = B67
6.  (B90) = {B67 + (B67) } ∧ A
7.  B90 = B90+1
8.  B69 = B90
9.  B67 = B67+1
10. B43 = (B67)
11. If B43 is &, that is F1 bit set then:-
            (B90) = B62
            (B90+1) = B63
            (B90+2) = (B65 + B43) ∧ A
             B90 = B90+3
             goto  9
12. If B67 = (B90-1) then recognition and goto 16.
13. B92 = (B62+1)
    B93 = (B92)
14. If B43 ≠ B93 then no match and
    either B90 ≠ B69 and we set:-  B90 = B90-3
                                   B62 = (B90)
                                   B63 = (B90+1)
                                   B67 = (B90+2)
                                   goto 12
    or B90 = B69 and we set  (B72-2) = B72-4
                              B64 = 0
                              B62 = 0
                  and exit with no match
15. If B43 = B69 then character is matched and we set B62 = B92 and goto 12.
16. Dictionary entry has been recognized:-
        (B63) = B43
         B63 = B63+1
         B90 = B68
         B64 = 1
         B69 = B61
         B68 = B62
```

```
17. CALL      R252   (Split chain into two sub-chains)
18. B61 = B69
    B62 = B68
    B63 = B88
    CALL R222  (Dual Routine)
```

The routine 252 splits the recognized and unrecognized part of the input stream into two sub-chains. On entry B69 points to the whole input chain and B68 points to the last character that has been recognized. On exit B68 points to recognized chained list and B69 points to the unrecognized part. If recognized part is null then B68 = 0 and if unrecognized part is null then set B69 = 0.

The routine 222 takes on analysis record pointed at by B63 and converts it to the Dual form. In the simple form of analysis record we have so far it is only necessary to move the one word analysis record to the top of the stack. To speed up the return mechanism the Routine 222 first moves the stack to its position before entry to 222 so that the return on exit causes control to go right back to the Master Routine. Orders required are:-

```
 B90 = B90-4
(B90-1) = (B63)
 B63 = B90-1
(B90) = 0
 B72 = B72-5
(B72-2) = B90+1
 Return
```

2.8. Item Assembly Routine

The ITEM assembly routine was originally designed to compile Atlas Machine Orders. On examining the Compiler-Compiler it was found that the set of orders used corresponded very closely with the set of orders required for manipulating the B variables and with very little change it was possible to remove the Atlas dependent features. It can, therefore, be thought of as an assembler for certain macros which describe operations on the B variables.

The B variable can be thought of as being split into several fields:-

J	a 3-bit field
Z	a subsidiary address field
S	2-bit field
A	main address field
F	2-bit field.

The Z-field need not be as long as the field A and its length is one of the parameters of a particular implementation. It should be at least 7 bits in length. It will be filled by first inserting the information in the A field followed by a move order which will move it to the Z-field. The contents of the Z-field will be extracted in a similar manner.

Logic operations are provided on all fields other than the Z-field.

The syntax of an ITEM routine is as follows:-

The symbol | is used to delimit alternatives and * and ? have their Compiler-Compiler meaning.

```
<Item Routine> ::= ITEM <N> <EOL> <instruction * ? >
<N> ::= an integer
<EOL> ::= newline
<instruction> ::= <label ?> <unl-instruction> <EOL>
<unl-instruction> ::= <H-instr> | <C-instr> | <I-instr> | <OP-instr>
<label> ::= / <N> /

<H-instr> ::= H = <H-addr>
<H-addr> ::= <N><+JSF * ?>
<+JSF> ::= + J <N> | + S <N> | + F <N>

<C-instr> ::= C = <C>
<C> ::= possible input character

<I-instr> ::= I<N>

<OP-instr> ::= <LOGIC> | <MOVE> | <NORMAL> | <TEST> | <TRANSFER> | <OUTPUT> |
               <101> | <121> | <124>
```

N.B. When Z,S fields are present then the A field need only be 10 bits in lengt

```
<LOGIC> ::= <LOGICOP> , <BA> , 0 , <LOGICADR> | 0165, <BA> , <BM> , <LOGICADR>
<LOGICOP> ::= 0127 | 0167 | 0121
<LOGICADR> ::= <N> | F <N> | J <N> | A | M
<BA> ::= integer between 0 and 126
<MOVE> ::= <MOVEOP> , <BA> , 0 , 0
<MOVEOP> ::= 0105 | 0106 | 0163 | 0164
<NORMAL> ::= <NORMALOP > , <BA> , <BM> , <XN>
<NORMALOP> ::= 0101 | 0104 | 0107 | 0110 | 0113 | 0114 | 0121 | 0122 | 0123 |
               0124 | 0145 | 0147 | 1102 | 1117 | 1166
<XN> ::= X <N> | <N>

<TEST> ::= <TESTOP>, <BA>, <BM>, <XN>
<TESTOP> ::= 0152 | 0170 | 0172

<TRANSFER> ::= <TRANSFEROP> , 127, 127, L <N>
<TRANSFEROP> ::= 0224 | 0225 | 0226 | 0227

<OUTPUT> ::= 1064, 0 , <BM> , 0 | 1064 , 0 , 0 , <N> |
             1065 , 0 , 0 , <N> | 1066 , 0 , 0 , <C>

<101> ::= 0101 , 127 , <BM, <XN>

<121> ::= 0121, 127, 127, 1 <N> | 0121, 127, <BM> , <N> |
          0121, <BA> , 127, L<N>
<124> ::= 0124 , 127 , 0 , 1 <N>
```

The semantics of these instructions are:-

2.8.1. <H-instr>

This sets the next store location equal to the value of the expression on the right of the equal sign.

2.2.2. <C-instr>

This sets the next store location equal to the numerical number equivalent to the character defined in the address field.

2.8.3. <I-instr>

This sets the index position <N> equal to the current position in the record store. It, therefore, provides a subsidiary entry point to an ITEM routine.

2.8.4. <OP-instr>

These correspond fairly closely to the numerically equivalent Atlas machine orders. (The <MOVEOP>'s and 1066 are different.)

In the address field we have:-

<N>	meaning the constant integer
X<N>	meaning the location of the Nth index position
F<N>	meaning F field set to N
J<N>	meaning J field set to N
A	meaning address mask
M	meaning mask for instruction word other than address field

The B-variable 127 on Atlas is also the location counter. The value of L<N< is the relative position of label <N> from the current position of the location counter. Therefore, 121, 127, 127, L<N> is equivalent to setting the location counter equal to its present value + the relative position of L<N> from the present value; that, jump to label <N>. Looking at it this way will perhaps show the reason for the different instructions and also some hint on how to implement them on a particular machine.

2.8.5 <LOGIC>

0127, i , 0 , k	sets $Bi = Bi \wedge k$
0167, i , 0 , k	sets $Bi = Bi \vee k$
0121, i , 0 , k	sets $Bi = k$
0165, i , j , k	sets $Bi = Bj \wedge k$

2.8.6 <MOVE>

0105, i , 0 , 0	Z field of Bi moved to A
0106, i , 0 , 0	A field of Bi moved to Z
0163, i , 0 , 0	S field of Bi moved to A
0164, i , 0 , 0	A field of Bi moved to S

2.8.7 <NORMAL>

0101, i , j , G	sets $Bi = (Bj+G)$
0104, i , j , G	sets $Bi = Bi + (Bj+G)$

0107, i , j , G	sets Bi = Bi $\wedge$ (Bj+G)
0110, i , j , G	sets (Bj+G) = (Bj+G) - Bi
0113, i , j , G	sets (Bj+G) = Bi
0114, i , j , G	sets (Bj+G) = (Bj+G) + Bi
0121, i , j , P	sets Bi = Bj+P
0122, i , j , P	sets Bi = Bi - Bj - P
0123, i , j , P	sets Bi = - Bj - P
0124, i , j , P	sets Bi = Bi+Bj+P
0145, i , j , G	sets Bi = Bj $\wedge$ (G)
0147, i , j , G	sets Bi = Bi $\vee$ (Bj+G)
1102, 70, 76, k	sets B70 to return; inserts k into store at this point and jumps to DOWN routine
1117, 0 , 0 , 0	terminates execution
1166, 67, 0 , 0	see TRANSPLANT

2.8.8. <TEST>

These instructions should be followed immediately by a <TRANSFER> instruction. They set the accumulator or some store to positive, negative or zero.

0152, i , j , G	sets accumulator = Bi - (Bj+G)
0170, i , j , P	sets accumulator = Bj + P - Bi
0172, i , j , P	sets accumulator = Bi - Bj - P

2.8.9. <TRANSFER>

Jumps to label defined in instruction depending on accumulator value.

0224, 127, 127, Li	jump to Li if Accumulator = 0
0225, 127, 127, Li	jump to Li if Accumulator $\neq$ 0
0226, 127, 127, Li	jump to Li if Accumulator $\geq$ 0
0227, 127, 127, Li	jump to Li if Accumulator < 0

The label position is defined by inserting (i) before the required instruction. Later it will be seen that all routines must be relocatable in the record transfer. For this reason all jumps should be made either relative to the entry point of the routine (B71) or relative to themselves.

2.8.10.<OUTPUT>

These instructions produce output on a selected device as follows:-

1064, i , j , 0	outputs character with numerical value Bj
1064, 0 , 0 , k	outputs character with numerical value k
1065, 0 , 0 , k	outputs newline (this could be extended)
1066, 0 , 0 , D	outputs character D.

2.8.11.<101>

This is an indirect transfer instruction

0101, 127, j , X_0	transfers control to the address given by the contents of Xj
0101, 127, j , G	transfers control to the address given by the contents of Bj+G

2.8.12. <121>

0121, 127, 127, Lj	transfers control to Lj
0121, 127, Bj, 0	transfers control to address stored in Bj
0121, Bj, 127, Lj	sets Bi equal to address of label Lj

2.8.13. <124>

0124, 127, 0 , Lj	transfer control to Lj

The complete action of the ITEM assembler can be described as follows:-

1. Get Item number and store current value of the record store pointer B88 in the corresponding Index position. Replace remains of ITEM line back on main chain. Set /B54 = Item Number and old Item address in X165.

2. CALL R238, R261, R215 as in steps 1. to 7. of Master Routine.

3. If B64 $\neq$/0 on return from analysis routine then we have recognized the next Master Phrase and so we must set B79 to point to complete line (i.e., rejoin recognized and unrecognized parts), complete any label references which cneed to be fixed and exit.

4. If B64 = 0 then no recognition and this must be the next instruction of the ITEM routine. Assemble and store in B88 region updating B88 so that B88 always points to the next free location in the record store. Replace line back on chain and repeat from 1.

2.9. ITEM Assembler

Although different implementations will be forced to use different methods for the assembler, it is probably worthwhile giving a broad description of the way the G-21 assembler is organized. A complete flow diagram of the routine is given in the flow charts marked ITEM ROUTINE 266 (i), (ii) and (iii).

The outer structure given above is shown on (i). To allow the DELETE ITEM routine itself to be deleted it is necessary to plant the address of the new routine in X_{13} initially, and only when the routine is complete will this be moved to the corresponding index position.

Two adjacent tables pointed at by B81 and B78 are required for handling labels. The first table pointed at by B81 has 40 entries corresponding to the addresses of labels 0, 1, 2, etc. The second table pointed at by B78 contains the addresses of all the instructions which referred to labels. These instructions are compiled initially with the label number in the address field. At the end of the routine this table is looked at and all the instructions have their correct address parts inserted.

2.9.1. CODE SKELETONS

On the G-21 it was decided to store in a table code pieces for all the possible instructions macros and pick out the required skeleton depending on the instruction. There are 32 instruction opcodes, and these can be stored economically and with quick access by having a 32 entry Hash Table having as key $(c \div 2) * 8 + d$ where the opcode is abcd. For example, the opcode 0152 has key $22_{(8)}$. ($5 \div 2 = 2$; $* 8 + 2 = 22$)

If two keys pointed at the same entry point then the key had 5 added to it and the process repeated.

The complete table used was:-

OP CODE	POSITION
0110	00
0101	01
1102	02
0113	03
0104	04
0105	05
0106	06
0107	07
1066	10
0121	11
0122	12
0123	13
0124	14
0225	15
0226	16
0127	17
1064	20
0224	21
0152	22
0114	23
0227	24
0145	25
1117	26
0147	27
0170	30
1065	31
0172	32
0163	33
0164	34
0165	35
1166	36
0167	37

This table then gives the position of an entry in two other tables. The first is used as a switch, depending on the type of opcode (shown as 'Jump on type' at bottom of flow diagram (ii)); the second points to the skeleton for the particular opcode.

There are basically two types of skeletons. The first is for the standard opcodes where the pointer is direct to the required skeleton. The second is for the 0101, 0124, 0121 type of instruction where the instruction may have one or more sub-forms. In this case the pointer is to a word containing the length of the first skeleton followed by that skeleton. This is repeated until we reach the last sub-form.

Each instruction has its skeleton divided into sub-skeletons grouped in the same way as the sub-forms are grouped. This is not necessary, as, of course, only one set of instructions is required for each opcode. However, in the cases where either the <BM> or address fields are zero, considerable economy of code may be achieved. Therefore, each skeleton is divided into four sections with the lengths preceding them:-

1.	$BM \neq 0$	$Address \neq 0$
2.	$BM = 0$	$Address \neq 0$
3.	$BM \neq 0$	$Address = 0$
4.	$BM = 0$	$Address = 0$

On the G-21 the address field of an instruction is not as large as the field of a B variable (as it is on Atlas). Consequently, several instructions have to be changed to alternative instructions when the longer field is required. For example, 0121 with a constant larger than the contents of the address field must be changed to 0101, together with the address constant stored away in an address. A typical example is:-

0121, 90, 0, F1

which would be changed to:-

0101, 90, 0, R

where R is an address containing the constant F1.

The standard flag settings are stored in an array in the bootstrap.

2.10. Machine Dependent Routines

It is intended that all parts special to a particular machine should be included in the bootstrap. The routines in this section at present are:-

275 DEFINE COMPILER <N>

This has been implemented on the G-21 such that the parameter defines the logical file that the compiler should be dumped on. It is necessary, before dumping, to pack up the dictionaries in the Record Store by a call of routine 243 and to reset the values of X_2 and X_3 to the current values of B88 and B87, respectively. It is then only necessary to dump the INDEX, Record Store and any constants used.

219 PRINT B82 IN OCTAL

248 PRINT B82 IN DECIMAL

Two routines which print the value of B82.

241 PRINT B VARIABLES

This routine is useful for debugging purposes and prints the values of all B variables.

283 PRINT FROM B125 to B126

Another debugging routine. The G-21 version dumps the area of store between value of B125 and B126.

216 INTEGER MULTIPLICATION AND DIVISION

As implemented on the G-21 and Atlas this is a non-standard routine which has its return address stored in B97 and has its two arguments in B98 and B99. If B97 has the F1 flag set it calculates B99/B98, otherwise B99 * B98. The result is returned in B99.

7 TRANSPLANT SKELETON

As will be described later, each statement used in the body of a routine is interpreted unless a compile-version of the relevant statement is available and the statement has simple parameters. The form of interpretation is to insert instructions in the routine which will call the TRANSPLANT routine and follow them by the analysis record for the particular instruction. On Atlas this can be achieved by a single instruction which is stored in X_7. This value is then picked up and inserted in the Record Store as desired. On the G-21 it was necessary to insert a set of instructions, and these were pointed at by X_7. The instructions must set B70 pointing two places before start of analysis record, then jump to address stored in B74 (entry point to transplant routine).

2.11. Conclusion

Once the above has been written for a particular machine it should be possible to write the remainder of the Compiler Compiler in a machine independent form. The next phase is to write a set of ITEM's which will overwrite the crude versions of the routines like the Analysis Routine already defined. In general, the initial versions of the routines need not be relocatable in the Compiler Compiler form as long as the order of changing routines is in the reverse order to the one in which they were defined. For this reason the ordering of the routines given above in the Record Store should be as follows:-

1. Machine dependent routines
2. 161, 214, 261, 215, 252 and 222, in that order.