INTERNATIONAL COMPUTERS AND TABULATORS LIMITED.

Programming Languages Division.

FORTRAN NOTE 26

June, 1967.

The 16K EDS Compiler #XFAE

This Note is intended to describe the Systems segments of the E.D.S. version of the compiler, insofar as these differ from the magnetic tape version. However, sections which are common to both versions are also outlined where this seems necessary for an understanding of the whole.

The material is divided into two parts, which are complementary.

The first part describes the various routines, while the second part contains a numbered list of names of areas, markers, pointers and so forth, together with notes on their use in the compiler. Any mention of an item which is described in the second part of the Note is indicated by the appropriate number in square brackets, enabling the item to be looked up directly.

Descriptions of Routines

1. Initial entry to compiler

Entry is at word 20 for paper tape input, 21 for card input and 22 for ATLAS card input. Other entries are at word 27, which causes all peripherals to be released and the computer to halt PF, and word 28, which simulates the detection of a FINISH statement,

All entry points are in segment FA2D.

When entered at 20,21 or 22 the compiler sets words 30 to one of three values depending on the entry.

The routine DISM is next called. This does nothing in the normal case since NUMIN[1] = 0. If however the compiler has been re-entered after an abortive compilation, NUMIN will not be zero and a console DISPLAY message will be typed.

Having read in the appropriate overlay, the compiler then enters it at the cue FMAS1.

2. FMAS1

This routine performs that part of the initialisation of the compiler which is required once only, at the beginning.

The following markers are zeroised:

ØVMMM[3], STINI [7], PDMK [8] SGMK [9], INPSW [10], and SCLM [12].

The following lower areas are zeroised:

LC1 common area; Part of LC3 common area; the first 511 words of LV.

The following initialisation is also performed:

ØVNDX[2] is set to #40000000 (indicating no overlay list yet)
PRIØ[4] is set to #500
CORE[6] is set to the number of words of store initially required by the compiler, minus 10.
MØNI[5] is set to 1 indicating assumption of TRACE level 1.
CMØD[11]: Bit 0 is set to 1.
Bits 1 - 23 are set to zero.

The routine PCH6 is called, to initialise the pointers CØBP[13] and SCØB [14]
The routine STINP is called to initialise TAPER [15]. TAPER is first zeroised (all 6 words). Bit 1 of word 30 is then tested to find out whether input is initially on cards or paper tape. The appropriate device is allotted by calling APER (described later).

The routine PLIE (outlined later) is called to read a line from the input device into the area CIMAGE[16].

The routine STSE (outlined later) is called to compare the contents of CIMAGE with a table of acceptable statements, setting X3 to a value depending on the results of the comparison. X3 is then used as a modifier to obey the appropriate instruction in a branch table. In this case, the only "acceptable" statements are LIST(TP),

SHORTLIST (TP), and NO LIST. The first two of these cause LISMK [17] to be set to 1 (tape punch); NO LIST causes it to be set to 0; anything else causes it to be set to 2, (line printer assumed).

Note: The first line remains in CIMAGE and is examined again during the routine FMAS2.

The routine STLST is called. This calls APER to allot a listing device if required, and prepares to start listing.

The routine PLST3 is called, causing the first line to be actually listed (if there is a listing device).

IND1 [18] is set no-zero before entering the routine FMAS2.

3. FMAS2

This routine performs that part of the compiler initialisation which has to be done before every segment is dealt with.

SØL [20] is set equal to the address in SØLØ [19] thereby causing the local lists for the segment to start at the beginning of the listspace.

The markers EOFM[21], SCLM[12], and EMARK[22] are zeroised.

If IND1[18] is zero, the routine PLIE (outlined later) is called to read a line from the input device into CIMAGE [16]. Otherwise a line is assumed to be already there.

The routine STSE (outlined later) is called to compare the contents of CIMAGE with a table of acceptable statements, setting X3 accordingly for use in obeying a branch table. Acceptable statements here are all intersegment statements, and any statement which introduces a Fortran segment or the Program Description. Anything else results in a branch to ER17, which flags an error 17.

FMAS2 is the usual return point after processing of intersegment statements or of whole segments.

4. SEGL1

The branch table used by FMAS2 results in a branch to SEGL1 for all statements which introduce Fortran Segments.

STIN1[7] and SGMK[9] are set non-zero.

The routine INE is now called. This completes the initialisation required for any Fortran segment, and includes the setting up of the 23 index words for the 23 threads of the main list. The moving list pointer NAVL[23] is set equal to SOL[20] initially. The local cue list index word is set up; the LC2 common area is zeroised; the routine PCH6 is called to initialise the output buffer pointers COBP[13] and SCOB[14]; and the first S/C record is begun by inserting the '3' needed for beginning a segment title block. A few other things, which have not been mentioned are also initialised.

SEGL1 finally branches to PRSL1 (see next section)

5. PRSL1

This routine constitutes the main path through the compiler for the processing of each Fortran statement.

When PRSL1 is entered, a line will already be in store waiting to be processed. However, the compiler has to find out whether there are any continuation lines, and to do this calls PSSE (outlined later). PSSE continues reading further lines until the first line which is not a continuation line is found, and builds up the statement in the area STATEMENT[24]

When the statement is complete the label field is dealt with. Discrimination as to type of statement is then carried out by calling STATE. This uses the routine STSE to compare the statement type with one of three tables of acceptable statements, setting X3 accordingly for use with a branch table. The table used for comparison is governed by the contents of SCLM[12]. If SCLM is zero only a statement which introduces a segment is acceptable; if it is 1, then any statement permissible is an ordinary Fortran segment is acceptable; if it is 2, only those statements permitted within a BLOCK DATA segment are acceptable. Intersegment statements are of course not allowed at all here.

Arithmetic statements are identified separately, and for these the central compiler is entered immediately.

If SCLM=1 (i.e. within an ordinary segment) or SCLM=2 (i.e. within a BLOCK DATA segment) the compiler branches straight to the appropriate part of the central compiler.

If SCLM is zero the only acceptable statements are MASTER, SUBROUTINE, BLOCK DATA, and the various types of FUNCTION statement. These are dealt with respectively by the routines MASE, SPSE, BLOCK, and (at least eventually) FPSE. In all these cases the first thing done is to set SCLM non-zero; it is set to 1 unless the statement is BLOCK DATA in which case it is set to 2. Having done that, processing of the statement itself begins, and that can reasonably be considered as part of the central compiler.

When a statement has been completely processed by the central compiler, control returns to CC which forms part of STATE.

The compiler then exits from STATE and continues through PRSL1. If the next statement (the first line of which will be already in store) is an END statement, it calls the routine ENDE (outlined later) to process it, and then branches back to FMAS2, since it is no longer within a Fortran segment. If the next line is not an END statement, control returns to the beginning of PRSL1 and the whole process is repeated for the new statement.

6. RPER

This routine deals with the release of peripherals or closing of files.

Calling sequence:

```
LDN  1  n
CALL 5  RPER
```

where n is the number of the relevant word in TAPER [15], and is therefore in the range 0 - 5.

Action: If the whole of the relevant TAPER word is zero, no device of that class has been allotted. In that case RPER merely exits. If the relevant TAPER word is non-zero, it is zeroised and the peripheral is released or the file closed as the case may be.

On exit, X1 and X7 will be the same as at the beginning. X7 sometimes contains useful information throughout.

Note: All release of peripherals or closing of files is done by this routine.

7. APER

This routine deals with the allocation of peripherals or opening of files, and all such operations are done by this routine.

Calling sequence:

```
LDX  7  'C/M'   (or equivalent)
LDN  1  n
CALL 5  APER
```

where n is the number of the relevant word in TAPER [15], and is therefore in the range 0 - 5. C/M is a counter modifier for the device or file required, in the form type/unit number, (B0 must be zero). This counter modifier is transferred by APER into bits 1-23 of the TAPER word. If the device is successfully allocated, or the file opened, bit 0 of the TAPER word is set to 1 before the exit.

In the case of a slow device the routine attempts to allot it by means of an ALLOT instruction; if successful it merely sets Bit 0 of the TAPER word to 1 and exits. If unsuccessful it halts TR, TP, LP, CR or CP as the case may be, and a further attempt can be made by typing GO.

In the case of magnetic tape or EDS the area MFIL[25] must have been previously set up as follows:

MFIL to MFIL+2: Name of file to be opened.
MFIL+3: File generation number for insertion in control area.
If the device is magnetic tape this information is transferred to the control area and the file is then opened.

If the device is EDS, words 1 to 8 of the control area are first zeroised to get rid of any reply information from previous operations. The file details are then transferred from MFIL to the control area and the file is opened. Finally the routine TRW1 (see next section) is called to test the reply word. If the reply is not satisfactory a halt occurs depending on the exact reason for this. A further attempt to open the file can be made by typing G0.

On exit from APER, X1 and X7 are the same as on entry. Also X2 contains the device type number and X3 the unit number.

Note: Whereas RPER can safely be entered even if there is nothing to release, APER must not be entered if the device concerned is already allocated.

8. TRW1

This routine tests the reply word after an attempt to open an EDS file. It uses X6 as the link and expects to find the reply in X2. If the reply is satisfactory it returns to link+1. Otherwise it returns to link+0 if G0 is typed after the halt.

Note: APER relies on TRW1 not to use any accumulators except X2 and X6.

9. PLIE

PSSE

PLNE

These are the three entries to the peripheral-independent part of the input system.

PLIE is entered when the compiler is between segments (or before the first segment) and wishes to input the next line. In that case PLIE will be called from FMAS2 and one of the following types of line would be expected:

(1) An intersegment statement
(2) A statement introducing a Fortran segment
(3) A type 3 semi-compiled record (or type +)

PLIE first sets SCIND[27] equal to zero then calls INPUT (described later) to read the next line into CIMAGE[16]. If the line is in fact semi-compiled (type 3 or +), this fact is detected before returning to PLIE and SCIND is set non-zero. In that case, since SCLM[12] is necessarily zero in these circumstances, PLIE immediately branches to the routine SEMI (described later). SEMI outputs the semi-compiled record, and calls PLNE. Since PLNE does the same as PLIE except that it does not zeroise SCIND, the next line is then input and the compiler branches back to SEMI. Semi-compiled is thus copied across to output until SEMI detects the end of the segment and branches to FMAS2 instead.

If the line read in is an intersegment statement, SCIND [27] is left zero by the routine INPUT, and the path followed through PLIE is governed by the fact that SCLM [12] and SCIND will be zero and V will be unset. This path eventually goes through PLSCX, where the end of line marker #74 is inserted in CIMAGE [16], to PLSL8 and exit. On exit from PLIE the line is compared with a table of acceptable statements by FMAS2, and the compiler then branches to process it. Exactly the same procedure is followed for lines which introduce source segments, since the statement is not identified until FMAS2 compares it with the table.

PSSE is entered from the routine PRSL1 (already described). This only occurs when the compiler knows it is dealing with a Fortran source segment. The first time it is called in a given segment, the line introducing the segment will be in store. However, the compiler must find out whether there are continuation lines always, and to do this calls PSSE. PSSE first sets V and zeroises SCIND[27]. SCLM[12] will be zero if this is the first time in the segment that PSSE has been called; otherwise it will be non-zero. Bearing these various points in mind, the path followed can be traced. Further lines are input (by PLS5, not by PLSCC which is by-passed when entry is at PSSE) until the first line which is not a continuation line is found. The complete statement is built up in the area STATEMENT[24]. The statement is then subjected to a "preliminary line scan" (PLSC7) which finds some of the most blatant types of error, and the compiler finally exits via PLSL8.

Note: Except when semi-compiled records are being dealt with, a pointer called PØSI [28] is set up before the exit. If entry was at PLIE, it is set to point to the seventh position in CIMAGE [16] which is where the statement itself is expected to start. If the entry was at PSSE, PØSI is set to point to the beginning of the area STATEMENT[24], which is the place where further processing will begin in that case.

10. INPUT

This routine forms the peripheral-dependent part of the input system. It is usually called in the course of PLIE, PSSE or PLNE (already described) but is also called or branched to from other places.

It first deals with the special case of Read From (ED,...) continuation lines by branching to TRSP8 if INPSW [10] is non-zero. Otherwise it space-fills the area CIMAGE[16], and then goes on to determine, by examining the contents of TAPER [15], which is the current input device. A slow peripheral will be present in all circumstances, but if TAPER indicates that EDS or magnetic tape is in current use also, the EDS or magnetic tape is selected rather than the slow device.
The compiler branches to one of four sections accordingly:

TRSPL:	Paper tape input
CRSPL:	Card input
EDSPL:	EDS input
MTSPL:	Magnetic tape input

TRSPL: The paper tape record is read initially into the area BUFF [29]. If SCLM [12] is zero, the routine SETSC is called, which sets SCIND [27] or leaves it unaltered as the case may be. The record is then moved to CIMAGE [16] after interpretation and removal of all shift characters. Finally NLIND [30] is set.

CRSPL: Cards are read into CIMAGE [16] directly. If SCLM[12] is zero, the routine SETSC is called, which sets SCIND [27] or leaves it unaltered as the case may be. If the card is in ATLAS code (ascertained from word 30 of store) it is translated into standard code in the CIMAGE area. The rest of the routine, from CRSP1 onwards is devoted to working out what should be stored in NLIND[30]. This last section is used at the end of MTSPL and EDSPL also.

EDSPL: The areas BEMPT[31], BKNUM [32], and BUCIN+4 [33] will have been initialized during the processing of the READ FROM (ED,...) statement. All EDSPL has to do is to continue inputting records from EDS until the requisite number of buckets, as indicated by BKNUM, have been dealt with. It does not therefore have to look out for an end sentinel of any kind. Extraction of records from successive buckets is accomplished with the aid of SCN1 [34] and SCN7[35]. The actual reading of buckets is done by the routine RBUCN. This also tests the reply word by calling TRW4 (described later) and sets SCN1 [34].

When BKNUM has been reduced to zero, the compiler exits via EDSP3 (see next section).

MTSPL: This deals with input of source and semi-compiled program (whether batched or unbatched) from magnetic tape. It is much larger than EDSPL because the formats are less simple and because the ends of subfiles have to be located by testing of sentinels rather than by a simple counter.

Blocks are read initially into the area BUFF [29] and records are then extracted one at a time into the area CIMAGE[16].

When the time comes to return to the slow input device the compiler branches to BRIN2 after releasing the tape or setting bit 0 of TAPER+1 [15] to zero. BRIN2 makes sure that the correct overlay is in store, and then branches to INPUT, thus effectively continuing the attempt to obtain the next line by reading from the slow device.

11. EDSP3

BRINP

When any READ FROM is first detected, INLINK [37] is zeroised. If the READ FROM applies to EDS, the Read From routine finds the position on EDS where the data starts, and knows from the directory how many data buckets there are in the subfile.

The main compiler is then entered by calling BRINP, whose first action is to store X0 in FMLINK [36] thereby preserving the link back to the Read From routine. It then tests INLINK [37] to see if it is zero. The first time it will be, and control passes to FMAS2. Thus the compiler begins reading EDS, expecting to find an intersegment statement or a statement introducing a segment.

Compilation continues from EDS until the requisite number of buckets have been read, in which case control leaves EDSPL via EDSP3. Now at this point X0 will contain the link back to PLIE or PSSE, which will have called INPUT for the purpose of reading the nect line. This link is stored in INLINK [37], and, after reading in the correct overlay, the compiler goes back to the Read From routine via the link preserved in FMLINK. The Read From routine ascertains whether any more subfiles are to be read from EDS. If so, it sets up BEMPT [31], BKNUM [32], and BUCIN+4 [33] ready to read the next subfile, and then calls BRINP again. After preserving the link in FMLINK once more, INLINK is tested and this time is non-zero. The compiler ensures that the correct overlay is in store and then branches to INPUT, thereby effectively continuing the search for the next line from where it left off, and ready to exit back to PLIE or PSSE having found it.

When there is no more EDS input to come, the Read From routine closes the EDS file, causing TAPER+5 [15] to be altered accordingly, and branches to BRINP. BRINP restores the link from INLINK and branches to INPUT after ensuring that the correct overlay is in store. The effect of this is to continue searching for the next line from a slow device, returning, via the link, to PLIE wher it is found.

12. TRW2

TRW3

TRW4

These routines test the reply word after an "extend" operation or a "read/write" operation on EDS.

TRW2 is used after an attempt to extend a file. It expects to find the reply in X7, and an irrecoverable halt occurs if the reply is abnormal. The routine is relied on not to use any accumulator except X7 and the link X6. (It uses X2 if the reply is abnormal but there is then an irrecoverable halt anyway).

TRW3 is used to test replies after a write instruction on unit 1. The reply is expected to be in BUCK+1. If entry is at TRW4, the reply must be already in X7: This entry is used for input files. The routine is relied on not to use any accumulators except X7 and the link X6, unless the reply is abnormal in which case an irrecoverable halt occurs anyway.

13. SEMI

This routine copies semi-compiled segments which are input to the compiler onto the output file. Its basic method of use has already been described in connection with the routine PLNE. It lists the name of the semi-compiled segment, moves it from CIMAGE [16] to the area pointed to by SCØB [14], check-sums it, and causes it to be output by calling INTR1 (described later). If the record it has just output is a terminating record (type 4 or comma) it branches to FMAS2. Otherwise it calls PLNE to obtain the next record, which it deals with likewise.

14. GFE1

GFN

Both of these routines are used for picking up the next field in an input record. GFE1 is used normally, but not when the next field is expected to be a file or subfile name. In that case GFN is used.

GFE1 is described in detail in Fortran Note 12.

GFN, like GFE1, expects POSI [28] to be pointing to the first character in the field. Leading spaces, if any, are ignored. Thereafter, any character other than a number, a letter, a space, %, or - is regarded as a field terminator. On exit, both X5 and TERM [38] contain this terminating character. If the terminator is the first character encountered (other than spaces possibly), then X4 is negative on exit, indicating that there is no field. Otherwise X4 is zero on exit. Also on exit, X2 is pointing to the first character beyond the terminator, and the area NAME [39] contains the field which has been picked up. GFN treats spaces (other than leading spaces) as significant, unlike GFE1.

15. INTR1

INTR2

This routine is called when the compiler has set up a complete semi-compiled record in the output buffer pointed to by SCØB [14] and wishes to output it. One of the two calls to this routine is made from the peripheral-independent output segment FQ2A; the other is made from SEMI (described earlier). The routine tests CMØD [11], and if this is non-zero, indicating either that there are errors or there is no output device anyway, avoids outputting anything and merely resets the pointers SCØB [14] and COBP [13]. If CMØD is zero the routine calls PCH1 (described later) which deals with the outputting to EDS of the S/C record, and also resets the pointers.

16. PCH1/PCH6

This routine is called by INTR2 in order to output semi-compiled information to EDS.

The output system works roughly as follows:

Semi-compiled is output to a permanent file in the form of a single data subfile. If a program is successfully compiled, bucket 1 of the output file should contain a header, a directory description (DD) record, and two subfile description records (SFD's) one of which describes the subfile containing semi-compiled, while the other describes any empty space remaining at the end of the file. This directory bucket is not output until the whole program has been compiled. Although in the present version of the compiler it is not possible for the directory to overflow the capacity of a bucket, it was originally thought that batches of programs would have to be dealt with, and so provision was made for overflow of the directory. This makes the output segment unnecessarily complicated.

As soon as the building up of a bucket in the output buffer begins, a bucket is reserved for it on EDS by storing the number of the first unreserved bucket in either BKN1[43] or BKN2[42]. CNT[44] always contains the number of the first unreserved bucket in the file.

Output of semi-compiled information is double-buffered, the buffers being BUFR2 and BUFR3[46]. These are pointed to alternately by NCØB[45].

When PCH1 is called, DIR[40] is set to zero, and a word count is appended to the front of the semi-compiled record. If the record is a type 3 semi-compiled record, then (unless this is the very first time, in which case the current buffer will be empty) WRT2 is called, which has the effect of writing away the current buffer and starting a new one. Otherwise WRT1 is called, which has the effect of adding the semi-compiled record to the current buffer if there is room, or writing it away and starting another one if there is not. WRT1 and WRT2 are described below.

Finally PCH6 resets the pointers SCØB[14] and CØBP[13] ready for the next semi-compiled record.

17 WRT1/WRT2

These routines expect the following information to be set up beforehand:

DIR = 0 for semi-compiled output; DIR non-zero for directory output.
X1 points to the area from which the record is to be taken.
X4 points to a location containing the number of the bucket on EDS reserved for the current output buffer.
X3 points to the current output buffer (i.e. BUFR2 or BUFR3)

The accumulators X1, X4 and X3 are not altered in the course of the routine; this is important because the routine calls itself at one point and branches to itself at another.

If there is room for the new record in the current buffer, and if entry was at WRT1, then WRT2 is not entered. The routine merely moves the record into the buffer setting X6 to point to its position there (X6 is used after the exit).

If there is not enough room in the buffer WRT2 is entered. WRT6 is then called in order to output the buffer to EDS. A new bucket is reserved, buffers are switched and the routine HEDR (see below) is called, to set up a header in the new buffer. The routine then branches back to WRT1, causing the record to be output into the new buffer.

18. HEDR

This routine sets up a header in the buffer whose start address is held in X3. The routine is relied on not to alter X3 or any other accumulator except X7. The link is X0.

19. WRT6

This routine actually writes buckets to EDS. If the file is already full, it will be extended by 80 buckets and LASTB[48] will be altered accordingly. After any file extension, the reply word is immediately tested by calling TRW2 (already described). Before writing a bucket the reply word for the previous write operation is tested. This method of working arises from the fact that semi-compiled output is double-buffered.

WRT6 is relied on not to alter any accumulators except those which it alters in the present version.

20. FERRR

This routine is entered from the error segment when the first error in a program has been flagged. It merely releases the output file by calling RPER (already described). The compiler will be prevented from trying to write further buckets by the fact that CMOD [11] will be non-zero from that time on.

21. Program Description

The Program Description segment is entered from FMAS2 on detection of a PROGRAM (, OVERLAY PROGRAM(, or SEGMENTS statement. The only difference between these at present is that the first two must be followed by the program name whereas SEGMENTS on its own is permissible in which case the compiler assumes a name of "NONM".

If the Program Description is not the first segment, an error 17 results because SGMK[9] will be non-zero. Otherwise the overlay list pointer OVLLL[49] is initialised by making it point to the beginning of the list-space (whose address is held in SOLO[19]).

Since the overlay list is held in the list-space area and grows when OVERLAY statements occur in the Program Description, the local lists for the Program Description segment must be accommodated elsewhere. They are in fact accommodated in the Polish List area, which starts at PNPI in common area LC3. SOL[20] is therefore initialised to that address in this case.

Various other indicators are set non-zero including STIN1[7], PDMK[8], SGMK[9], and EMARK[22]. The usual segment-initialisation routine INE is also called, and the program name is stored for safe keeping. The PROGRAM, OVERLAY PROGRAM or SEGMENTS line is then processed (the cue list is begun etc).

PDCCE is the return point for all Program Description statements except the END statement. In PDCCE, the next line is read by calling PLNE (already described), and the compiler then compares the line with a table of acceptable statements and branches accordingly.

If in the course of the Program Description OVERLAY statements are detected, OVMMM[3] is set non-zero and entries are made in the Overlay list.

22. ENPD

When the END statement is finally detected in the Program Description the compiler branches to ENPD. If OVMMM[3] is non-zero, an entry for the overlay package %EROL is made in the cue list. An entry is also made for either %FERROR or %FERLIM depending on the contents of MONI[5]. CMOD[11] is then tested to find out whether any errors have been flagged. If not the routine ENDE (outlined later) is called. This outputs the leader for the Program Description segment. Having done that the Overlay list is no longer required and can be got rid of by setting @VNDX[2] to #40000000 again. Finally EMARK[22] is set zero, indicating the end of Program Description processing, and control returns to FMAS2.

23. ENDE.

This routine processes END statements and outputs segment leaders. The processing is somewhat different for END statements in Program Description segments to that for other END statements. EMARK [22] is non-zero if the Program Description is being dealt with, and zero otherwise. If errors have been flagged previously, as indicated by CMOD[11], most of the processing is omitted and control returns to FMAS2 almost immediately.

24. STSE

This routine is used in connection with the various comparison and branch tables used for discriminating between the various types of statement. It is described more fully in Fortran Note 12.

25. Fortran List System

This is described in detail in Fortran Notes 11 and 12. It must be remembered however that in the EDS Compiler there is no Consolidated Cue List, and that the Overlay list can be dispensed with as soon as the Program Description leader has been output. This means that all local lists (except that for the Program Description, which is held in the Polish List area) can start at the beginning of the list-space.

26. READ FROM

The Read From as applied to cards, paper tape and magnetic tape is essentially the same as in the magnetic tape compiler, except that a statement of the form READ FROM (MT,-. SRF5) is not allowed and results in the flagging of an error.

Any READ FROM detected while reading magnetic tape or EDS is ignored.

Any READ FROM causes INLINK [37] to be zeroised although this is irrelevant unless it is a READ FROM (ED,....).

All READ FROM's except the READ FROM (ED,...) eventually call RDFR to carry out the actual switching of peripherals. The calling sequence for RDFR is:

```
LDX  7  'C/M' (or equivalent)
LDN  1  n
CALL 0  RDFR
```

where n is the number of the relevant word in TAPER [15] and C/M is a counter modifier in the form Type/Unit no. This calling sequence is basically the same as that for APER (already described) and is in fact used by RDFR for calling APER.

READ FROM (CR/S1900) :
READ FROM (CR/ATLAS) : Bit 7 of word 30 is unset or set respectively. Otherwise these are treated in the same

way as READ FROM (CR), which merely sets up X1 and X7 and calls RDFR.

READ FROM (TR) is treated in a similar way.

In the case of a READ FROM (MT,...) the area MFIL[25] has to be set up. Details are given in connection with MFIL. Having done that, X1 and X7 are set up and RDFR is called.

Action of RDFR: If the device requested is a slow device, RDFR calls RPER to release the current device and APER to allot the new one. It then exits. If the device is magnetic tape and the first word of MFIL[25] is zero, RDFR expects a tape to be already allotted. Otherwise it opens one, taking the file name from MFIL. In either case it now tests word 1 of TAPER[15] and flags an error if this is zero. It then sets Bit 0 of TAPER+1 equal to 1. If MFIL+4 is zero, indicating a simple file, SIC[50] and SIMF[51] are set non-zero. SCT[52] is also set to zero before the exit. If MFIL+4 is non-zero, SIMF [51] is set to zero and the compiler scans the magnetic tape looking for the requested subfile, and positions the tape ready to begin reading.

On exit from RDFR control returns to FMAS2.

READ FROM (ED,....) : On EDS, unlike on magnetic tape, subfile names cannot be relied on to be unique, except within a particular directory subfile. It is therefore necessary to be able to follow a definite path down to a particular subfile, by specifying one subfile name for each level in the structure down to the level of the subfile required. The subfile is then located by following the appropriate pointer at each level.

A necessary consequence of this is that the READ FROM must contain a file name and an indefinite number of subfile names, the first of which applies to level 0, the second to level 1 and so forth. For this reason continuation lines are permitted as a special case, but subject to certain rules. These are that the first line must contain at least one subfile name, that lines may only be broken before a full-stop, and that no interspersed blank or comment lines are allowed. The format is therefore, for example,

```
READ FROM (ED,Filename(G),S/F1(G).S/F2(G)
1.S/F3(G).S/F4(G))
```

The generation numbers are all optional. If the generation number for the file is not specified, Executive assumes that the file with the highest generation number is required. If the other generation numbers are omitted, it is assumed that only one subfile of that name is present.

As the READ FROM is of indefinite length, it is not possible to read it all in before beginning to act on it. Thus as soon as the file name is known, an attempt is made to open the file, and as each subfile name is detected, the directory is scanned to find it. If an error is detected in the Read From after the file has been opened, it is released again at once so that input can continue from the slow peripheral.

Scanning of directories is carried out by the routine called SCAN. This expects to find the first bucket of the directory subfile in the area BUFF[29], and also expects the number of useful words in this bucket to be stored in SCN1 [34]. If the subfile name is found, the bucket containing it is left in BUFF, while SCN3 [53] points to the beginning of the required SFD.

When continuation lines are present, the difficulty arises that the slow peripheral must be re-activated to read in successive lines and also that the information currently in BUFF will be overwritten if the slow device happens to be a tape reader.

To avoid these difficulties the contents of BUFF are stored temporarily in the list space area (which contains nothing useful between segments), while the slow device is re-activated by removing bit 0 of TAPER+5[15]. The routine INPUT is then called to read the next line in (without listing it yet), and the end of line mark is afterwards inserted (INPUT itself does not do this; it is usually done in PLIE). The contents of BUFF and bit 0 of TAPER+5 are then restored.

When the SFD for the last-mentioned subfile has been located, the compiler branches to RDF13, where it prepares to find and read in all data subfiles implicit in the SFD. That is to say, the last-mentioned subfile may be composite, in which case it is assumed that all data subfiles in the structure based on it are required. To this end the compiler works systematically through the structure. Whenever a data subfile is located, the number of its first bucket is placed in BUCIN+4 [33], the number of buckets in it is placed in BKNUM [32] and BEMPT[31] is set to zero. The compiler then calls BRINP (already described) and returns only when the whole subfile has been input.

When everything implicit in the READ FROM has been input, the file is released and the compiler branches back to BRINP.

Further note on continuation lines: A continuation line is expected if the previous line appears to the compiler to be incomplete. If an expected continuation line is not forthcoming, the <u>previous</u> line is flagged as an error, and the new line is then treated as a new statement. For this reason, i.e: in case an intervening error flag has to be output, the continuation line is not listed as it is read in. INPSW[10] is used in this connection.

27. <u>SEND TO</u> (SNTBS)

This section is rudimentary at present because the compiler accepts only one form of SEND TO, and assumes that no output is required if no SEND TO is given.

The acceptable form is

SEND TO (ED,File Name(G).Subfile name(G)) where the generation numbers are optional. The compiler always sets up a new composite file in the file specified; consequently anything initially in the file is destroyed.

Bucket number 1 (the directory) is not output until the whole program has been successfully compiled. This is done by the routine CLE01 (see below).

If this is not the first SEND TO detected, then STIN1[7] will be non-zero. In that case the new SEND TO is ignored and control returns to FMAS2. If STIN1 is zero, it is set non-zero at this point.

The file name and generation number are stored in MFIL to MFIL+3[25] and the subfile name and generation number in MFIL2[26]. If no errors are detected, bit 0 in CM0D[11] is set to zero and the file is then opened by calling APER (already described). The number of buckets in the file is then determined and stored in LASTB [48].

Bucket 1 is reserved for the directory by putting 1 in BKN1[43], and Bucket 2 is reserved for the first bucket of S/C by putting 2 in BKN2[42]. BCNT[54] is also set to 2. CNT[44] is set to 3.

A header is then set up in the first output buffer (pointed to by NC0B[45]), and control returns to FMAS2.

28. FINISH

An Error 36 is signalled if PDMK[8] is zero, indicating that no Program Description was supplied. CM0D[11] is then tested to find out if there is an output device; this will be the case only if CM0D is zero. If there is an output device, the routine CLE01 (see below) is called. This outputs the directory. The compiler then halts ZZ or EC depending on whether there are errors or not.

29. CLE01

The current output buffer is first written away by setting up the control area and calling WRT6. The routine then prepares to output the directory. The link is stored in DIR[40] thereby both preserving the link and setting DIR non-zero as in necessary for outputting directory information. X1 and X3 are then set up in the manner required by the routines HEDR and WRT1 (already described). HEDR is called, causing a header to be set up in the buffer. WRT1 is called causing a DD record to be moved into the buffer, after which the word "COMPOFILE" is placed in its name field. X3 will still contain the address of the buffer at this point. X4 and X1 are set up also as required by WRT1, which is then called to output an SFD for the semi-compiled information. The subfile name and other details are then inserted in this SFD. Finally another SFD is output, covering any unused buckets left over at the end of the file. The buffer itself is finally written away by calling WRT6.

List of Names of Areas, Markers, Pointers, etc.

1. NUMIN Contains the number of instructions so far compiled. This is output as part of a DISPLAY message on the console at the end of a program by the routine DISM.

2. OVNDX Overlay list index word. This is a counter-modifier pointing to the last item in the overlay list. If the overlay list is empty, OVNDX is set to #40000000

3. OVMMM If non-zero, indicates that at least one overlay statement has been found in the Program Description.

4. PRIO Indicates the priority of the object program. Is set to #500 (indicating priority of 50) if no PRIORITY statement is given.

5. MONI Indicates the TRACE level. Is set to 1 by default.

6. CORE Contains the number of words of core store occupied by the compiler, minus 10. If the list space becomes exhausted, the compiler attempts to obtain more core and adjusts CORE accordingly.

7. STIN1 If non-zero causes any SEND TO statement which occurs to be ignored. It is set non-zero when a SEND TO has been detected already, or when the start of any segment has been detected.

8. PDMK If non-zero, indicates that the Program Description segment has been previously detected.

9. SGMK If non-zero, indicates that a segment of some kind (Program Description or Fortran) has been detected previously.

10. INPSW If non-zero, indicates that a READ FROM (ED,...) continuation line is being processed.

11. CMOD Bit 0 : Set to 1 if no output device (EDS) has yet been allotted.

Bits 1-23: The number of errors so far flagged by the compiler.

12. SCLM SCLM=0. Compiler is not currently dealing with material inside a Fortran source segment.

SCLM=1 Compiler is currently processing a Fortran source segment, other than a BLOCK DATA segment.

SCLM=2 Compiler is dealing with a BLOCK DATA segment.

13. COBP "Current output buffer pointer". A counter-modifier pointing to the area into which the compiler places semi-compiled records to be output. It is a moving pointer, used to insert successive fields; it starts at 67/BUFF1+1.

14. SCOB "Start of current output buffer". Points to the same area as COBP above, but is fixed and always points to the start of the semi-compiled record, i.e. to BUFF1+1.

15. TAPER A six-word area in the common block CSLCOM, which controls the use of peripherals at compile time.

Word 0 : Unused at present. It was intended to use it for an EDS scratch file as part of an improved method of peripheral switching (see separate write-ups). There is no reason why this should not be implemented eventually.

Word 1 : Magnetic tape input
Word 2 : Slow input (card or paper tape)
Word 3 : Listing device (line printer or tape punch)
Word 4 : EDS S/C output (permanent file)
Word 5 : EDS input (permanent file)

Each word of TAPER contains information as described below.

1) Whole word zero : No device of this type is currently allotted.
2) Word non-zero:
Bit 0:

Words except word 2 : Bit 0 is equal to 1 if the device is currently allotted to the program.
Word 2 : Bit 0 is equal to 1 if the device is currently allotted and is being actively used. If bit 0 is zero, the tape is merely being retained in case it is required again later.

Bits 1-8: Device type (e.g.: 5 for magnetic tape.)
Bits 9 -23:Unit number.
Device types must be as follows: Word 1 may only be 5. Words 4 and 5 may only be 6. Word 2 may be 0 or 3. Word 3 may be 1 or 2.
Unit numbers are as follows: Magnetic tape input is unit 2, slow input is unit 0, listing device is unit 1, EDS output is unit 1, and EDS input is unit 2.

All allocation and release of peripherals is carried out by the routines APER and RPER . Nevertheless, words in TAPER are sometimes altered temporarily in other places for special purposes, as far instance when dealing with READ FROM (ED,.....) continuation lines.

16. CIMAGE An 80 word area in common block LC0 into which individual lines of input are placed in the form of a "card image" (i.e. with shift characters removed etc).

17. LISMK Listing device indicator. LISMK=0 means no listing device; LISMK=1 means tape punch; LISMK=2 means line printer.

18. IND1 Set non-zero by routine FMAS1, before entering FMAS2. FMAS2 tests it and then zeroises it. If it was non-zero, FMAS2 avoids inputting a new line because the one input by FMAS1 has not yet been fully processed. Normally, of course, FMAS2 is not entered from FMAS1 but from somewhere else.

19. SOLO A constant in LP containing the start address of the compiler's list space. The list space is in Upper Common Variable block UC1.

20. SOL "Start of List". No consolidated cue list is built up by the EDS compiler; consequently the local lists for each segment can always start at the beginning of the list space, whose address is held in SOLO [19]. SOL is therefore initialised to this address by the routine FMAS2.

21. EOFM. "End of Fortran Marker." Believed to be redundant, but its complete removal would involve a change to the error segment of the compiler, which tests it.

22. EMARK. If non-zero, indicates that the Program Description segment (or its leader) is currently being dealt with.

23. NAVL. Local lists pointer (not fixed) which points to the location immediately after the last item in the lists.

24. STATEMENT. An area 331 words long in which Fortran statements (including continuation lines) are built up before being processed by the central compiler. Intersegment statements are not processed in STATEMENT however.

25. MFIL. An 8-word area in the common block CSLCOM, used for several purposes connected with the storage of file and subfile names taken from READ FROM and SEND TO statements:-

(1) READ FROM magnetic tape:
MFIL to MFIL+2 : File Name (MFIL set zero if none specified)
MFIL+3 : File generation number (zero if unspecified)
MFIL+4 to MFIL+6 : Subfile name (MFIL+4 set zero if no name specified, indicating simple file).
MFIL+7 : Unused at present but will be needed if subfiles are given generation numbers on mag. tape in future.

(2) READ FROM EDS:
The READ FROM (ED,...), unlike the READ FROM (MT,...), is acted upon as each field is read in. This is necessary because the EDS 'READ FROM' is of indefinite length. However this enables the first four words of MFIL to be used first for the File name and generation number, and then for all subfile names and generation numbers in succession, since each piece of information has been acted upon before it is over-written by the next field. MFIL+3 receives the file or subfile generation number if present; if none is specified it is set negative. MFIL+4 to MFIL+7 are not used in connection with EDS.

(3) SEND TO statements:
MFIL to MFIL+2 : EDS File Name
MFIL+3 : Generation number (set negative if not specified).
MFIL+4 to MFIL+7 : Not used. The subfile name and generation number from a SEND TO statement are stored elsewhere in the area MFIL2 [26]. It has to be preserved until the end of the compilation to be output in the directory; if it had been stored in MFIL+4 to MFIL+7 it would be in danger of being destroyed by the processing of a READ FROM (MT,...) statement.

26. MFIL2.

A 4-word area used for storing the subfile name and generation number from a SEND TO statement.
MFIL2 to MFIL2+2 : Subfile name
MFIL2+3 : Subfile generation number (set negative if not specified).

27. SCIND. "Semi-compiled indicator".

Set equal to 2 if a type 3 semi-compiled record is fed in. Remains equal to 2 until the compiler is about to input the first line following the type 4 terminating record; SCIND is then zeroised by the routine PLIE, which is called from FMAS2.

SCIND is set equal to 1 if a type + semi-compiled leader record is fed in, and remains so until after the type (comma) terminating block, when it is again zeroised by PLIE.

Except when reading semi-compiled, SCIND is zero.

28. POSI

A moving pointer which normally points to the next character to be examined in a field. The input routine (PLIE or PSSE) sets it initially to point to the first character of the statement which has just been input. In the case of PLIE this is the 7th character of CIMAGE [16]; in the case of PSSE it is the beginning of STATEMENT [24]. POSI is used by the routines which analyse input fields, namely GFE1 and GFN.

29. BUFF

A 128-word buffer in the common area AREA. Paper tape, magnetic tape and EDS blocks are read into it initially and then transferred, a record at a time, to CIMAGE [16]. It is also used for processing directory buckets in an EDS input file.

30. NLIND

A character counter/modifier which is set to point to the character after the last significant character which has been read into CIMAGE [16] by the routine INPUT. The counter part is set equal to 72-n where n is the number of significant characters present. The character address part is in the form 1.m starting with 0.0 for the first character position in the area.

31. BEMPT

Used when inputting data from E.D.S. It is set zero when the compiler is ready to read in the first bucket of data, indicating to the routine INPUT that no data bucket is yet in store. As soon as the first bucket has been input, it is set non-zero, so that the compiler will not read a new bucket each time it wants to input a record.

32. BKNUM

A decreasing counter containing the number of buckets in an EDS input file, which have not yet been input.

33. BUCIN

Control area for EDS input. BUCIN+4 is incremented directly whenever the next bucket is to be read.

34. SCN1

The bucket-input routine RBUCN sets SCN1 equal to the number of useful words in the bucket whenever a bucket is read from EDS. It is never set to any number greater than 128.

35. SCN7

Used in the EDS input routine. Points to the position immediately beyond the end of the present record in the input buffer, taking the first record to start at position 2.

36. FMLINK

Used for storing the link to the Read From (ED,.....) routine when the compiler is reading from EDS.

37. INLINK

Used for storing the link to PLIE or PSSE when the compiler returns to the Read From (ED,...) routine to find out if there are any continuation subfiles to be input. Set zero on detection of a READ FROM.

38. TERM

On exit from the field analysis routines GFE1 and GFN, TERM always contains the field terminator.

39. NAME

An 11-word area used for holding fields picked up by the field analysis routines GFE1 and GFN.

40. DIR

Used in connection with the output routines. Set zero if a non-directory record is being output. Set non-zero if a directory record is being output, it is sometimes used to hold the link in that case.

41. BLENG

Holds the length of a semi-compiled record as set up by the compiler.

42. BKN2

Holds the bucket number reserved for the next s/c bucket to be written.

43. BKN1

Holds the bucket number reserved for the next directory bucket to be written.

44. CNT

Holds the bucket number of the first un-reserved bucket on the output file.

45. NCØB

Output of S/C buckets (though not of directory buckets) is double-buffered. NCOB points to the buffer which is currently being built up.

46. BUFR2/BUFR3

The two output buffers (128 words each) one of which is pointed to by NCØB [45] at any time. Writing of S/C buckets takes place alternately from these.

47. BUFF1

A 21-word area. The compiler puts semi-compiled records into it starting at BUFF1+1. The output routine puts a word-count on the front before moving it to the output buffer. BUFF1+1 is always pointed to by SCØB [14].

48. LASTB

Contains the bucket number of the last bucket in the file. It is set up initially during the SEND TO routine, but may be altered by the routine WRT6 in the event of file extensions being necessary.

49. OVLLL

Overlay list pointer. This starts by pointing to the beginning of the overlay list, but moves as the list grows, such that it always points to the word after the last item in the list.

50. SIC

If non-zero indicates that the magnetic tape input file contains semi-compiled rather than source. The semi-compiled may be on a simple file or a may be in a subfile on a composite file.

51. SIMF

If non-zero indicates that the magnetic tape input file is a simple file. A simple file can contain semi-compiled only.

52. SCT

Indicates the depth within the subfile structure of the magnetic tape subfile currently being dealth with.

53. SCN3

Used when inputting from EDS. Points to the beginning of an SFD in a directory bucket relative to the start of the buffer BUFF[29]

54. BCNT

Set equal to the first bucket of semi-compiled information on the output file. It is used at the end of the program for calculating the number of buckets in the subfile for insertion in the directory.

P.M.Girard