SECTION 5 - SYNTHESIS AND RECORDING PROGRAM

5.1. General

The purpose of this section is to explain how an EMIDEC 1101 program is written, how this written program is translated into a medium suitable for feeding to the computer (i.e. punched paper tape or punched cards) and to consider the process through which instructions go as they are read by the computer and built up, in the computer store, into a complete program.

But let us first consider briefly what is happening inside the computer as it is carrying out a commercial task. There must be a program of instructions in the computer's store controlling each operation performed by the computer and as this program may well run to over a thousand instructions it cannot be held in its entirety in the 1024 word core store. Consequently the program is stored away on the magnetic drum and units of it are called into the core store as required, and it is from here that the instructions are selected sequentially and executed. The units of program being executed in the core store include instructions to call further units from the drum when they are required. Each of these units of program is called a 'Stage'.

The computer program is operating upon data read from magnetic tape, paper tape or cards. Some of the data is stored throughout the computer job so that the program can always refer to it e.g. price-lists and items descriptions, while other data is of a transitory nature, e.g. quantities of items ordered. Units of the store, both core and magnetic drum, are allocated to hold these various types of data, the sizes and positions in the store of these units depending only upon the special requirements of the particular task being carried out by the computer. Units of the computer store used by the program for storing data are called 'Sections'.

Watching a computer in action we can see data being read, we can see it, in binary, on the displays on the monitor desk as it is stored away in the Sections of the computer store and we can see results being printed, or carried forward on magnetic tape. So much for the data - we can readily appreciate how this gets into the computer, but what about the program, how does that get into the store?

The answer to this question is that programs are written, in Arabic numerals, in such a way that the individual instructions can be punched into paper tape or cards and fed to the computer as though they were data. It requires, of course, a program already situated in the computer to read this

data; a program with the basically simple task of reading and storing the data on the drum. This program, which we will for present convenience consider to be permanently stored in the computer, is called the Synthesis Program. While the Synthesis Program is controlling the computer in this task of reading a second program from paper tape or cards, there can be no question of the computer executing the instructions of the second program for they are being treated as just a series of numbers which have to be stored in specified locations on the drum. When the whole of the program has been stored, it is the last task of the Synthesis Program to transfer control to the newly stored program so that the latter can then take command.

In practice the Synthesis Program is designed to do much more than this simple reading and storing operation for, although the writing of EMIDEC instructions is quite simple, considerable thought is required to ensure that the store is suitably allocated to hold program and data and in organising the stages of program so that they can be called down from the drum to the core store when required. The extra work the Synthesis Program is made to perform simplifies the programmer's task, by giving him assistance in organising a program, making the alteration of existing programs a quicker and less error prone operation, reducing the amount of simple arithmetic he is called upon to perform, and allowing him to write instructions in a more convenient form than that required by the computer.

The Synthesis Program, like any other operational program, requires to have data presented to it in a standard form, and we will now consider how this data - the new program - must be written in order to be correctly interpreted and processed.

5.2. Writing EMIDEC Instructions

It will be remembered that a single EMIDEC instruction consists of five elements:- a function, 'a' address, 'b' address, 'c' address, and modifier. The 'c' address is further divided into two parts known as 'C1' and 'C2' addresses.

Inside the computer, instructions are expressed as 36 binary digits, and are stored one per word. The 36 bits are divided into groups to accommodate the elements of the instruction as follows:-

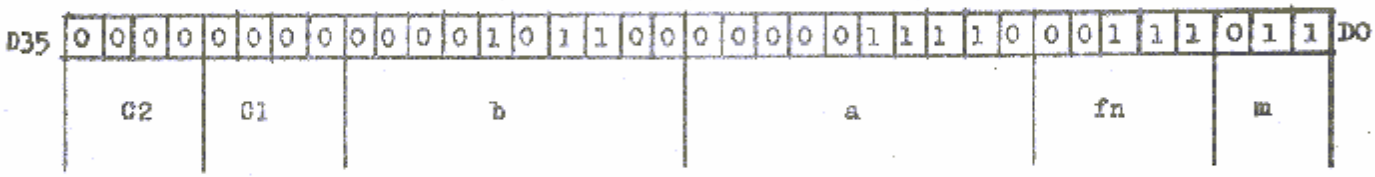
35 - 32	31 - 28	27 - 18	17 - 8	7 - 3	2 - 0
C2	C1	'b' address	'a' address	Function	Modifier

In writing instructions, however, it is more convenient to think in the order:- function, 'a' address, 'b' address, 'c' address, modifier. EMIDEC coding sheets are therefore drawn up in this order, with the aim of making program writing as simple as possible.

Take for example an instruction meaning "add the contents of register 30 to the contents of register 44, placing the result in register 44, after first modifying the instruction by the contents of register 3". This is written:-

	F		'a'	'b'	'C1'	'C2'	'm'
	7		30	44			3

For this instruction, after punching and reading in, it is only necessary to re-arrange the elements before storing in final form. After re-arrangement therefore, it would appear in the computer:-



One of the tasks of the synthesis program is to effect this re-organisation of instruction elements. Before considering this however, let us examine the Relative Address technique which is used in writing programs since it is necessary for these addresses to be recognised when the elements of an instruction are being read in.

5.3. Relative Addresses

(a) Data Storage

The example given above illustrates how the addresses of the operands to be used in an instruction are specified, both in the written form of the instructions, and as they are stored in the computer. In this example the numbers written in the 'a' and 'b' elements of the instruction are the numbers of the registers to be referred to. In order to write a complete program specifying in this fashion, the actual registers to be used, it would be necessary beforehand to work out the complete storage allocation plan. This is virtually impossible, and any later amendments to the program or the allocation plan would necessitate changes throughout the program already written. This would be a very inflexible method of writing programs, and would result in unnecessary errors and waste of time.

A method of writing programs has therefore been adopted which avoids the need to specify absolutely the registers to be used in each instruction. In this system, the programmer need only decide initially that he will use for data storage a number of sections of the store (of undetermined size initially), each designated by a letter of the alphabet A, B, C etc. Within each Section, registers can be numbered 0, 1, 2, etc. upwards; thus register 3 within Section B can be referred to as register B3. In writing addresses in this fashion, the Section letter, known as an 'indicator', is written in the faint-ruled column of the 'a' and 'b' address sections on the program sheet. The following is an example of an instruction so written:-

	1	A	25	D	6			

Addresses specified in this way are known as "Relative addresses", as opposed to "absolute addresses" when the actual register number is specified.

The computer can only obey an instruction when the addresses of the operands are expressed in absolute form; it is therefore necessary for all relative addresses to be converted to absolute addresses before the program is stored in the computer.

The conversion from relative to absolute addresses is obviously a very simple process for once it is known where, in the computer store, each section is to start, the absolute address of this starting point is added to the relative address written on the program sheet. Thus, with the example:-

	1	A	25	D	6			

if the programmer finds that in allocating store space to program and data it is convenient to make section A start at register 250 and section D start at 310 then location 25 of section A can now be identified as register 275 (absolute address) and location 6 of section D as register 316.

The programmer need only decide upon the section starting points when he has written the complete program and then, by providing the Synthesis Program with this information for each section, the conversion of relative addresses as written into the absolute form required by the computer is carried out as the new program is being read from paper tape or cards. Should amendments to a program extend to the re-positioning of sections of data in the computer store it is necessary only to re-state the new starting points of each section and no amendment is required to the individual instructions as written.

Relative addressing of Sections may be applied to both the core store and the drum store. In the latter case, each Section will comprise a series of 4-word blocks, since the drum is always addressed thus.

(b) Program Storage

The Relative Address system so far described only deals with data storage. It is in fact also necessary to use a relative address system for specifying the addresses of instructions in the core store. Every program is subdivided into a number of Stages, the maximum size of a Stage being 64 instructions. Within each Stage, instructions are numbered from 0 upwards on the program sheets. Any instruction in the complete program as written may therefore be identified by its Stage number, and instruction number within the Stage.

This becomes important when "test and jump" instructions are used in the program; again the computer can only act upon addresses expressed in absolute form, which means that in such instructions, the number of the register containing the next instruction to be obeyed must be specified. But the programmer cannot normally decide which registers to use for instructions until the program is completed, and the solution is therefore to specify instruction addresses relatively, by Stage number and number within the Stage.

The method of writing instruction addresses in 'jump' instructions is as follows:-

(i) the letter 'R' is used in the indicator column for all instruction relative addresses

(ii) if the program jump is from one instruction to another within the same Stage the instruction number within the Stage is written in the 'b' address column, thus:-

	F	'a'		'b'		C1	C2	m
	11		10	R	53			

The interpretation of this is "test whether register 10 contains zero; if so, the next instruction to be obeyed is in location 53 of this Stage".

(iii) if it is required to specify a jump to an instruction in another Stage, it is necessary to specify the new Stage number additionally. The two right hand (or least significant) digits specify the instruction number within the new Stage,

and the Stage number is written immediately to the left of them, thus:-

	F	'a'		'b'		C1	C2	m
	11		10	R	553			

meaning "test whether register 10 contains zero; if so the next instruction to be obeyed is in location 53 of Stage 5", or again:

	11		10	R	403			

meaning "test, and jump to instruction in location 3 of Stage 4". It is important to note that two digits must always be given for the instruction number when another stage is specified - in the example above instruction 3 is written as 03.

5.4. Punching New Programs

The Synthesis Routine will accept programs punched into cards or into paper tape. In either case the same rules apply when writing the program, including 'dummy functions' described later (5.6, 5.7, 5.8, 5.9).

Punched Cards

All instructions (including instructions with dummy functions 90-96) are punched in the same layout. Three instructions are accommodated on a card, and a field of 20 columns is allotted to each, divided as follows:-

Function	2 columns
A address indicator	1 "
A address	5 "
B address indicator	1 "
B address	5 "
C1 address	3 "
C2 address	2 "
Modifier	1 "

The three instructions occupy columns 1 - 60 of the card. In columns 61 - 65 is punched a number which allows the Synthesis Routine to check that the cards are in their correct sequence. This number consists of the stage number (3 columns) and the number within the stage of the third instruction on the card (2 columns); in the case of part-filled cards, the number should be punched as though all three fields were occupied. The index of sections and stages is punched with sequence numbers for 'stage 0'.

The beginning of a new stage, marked by dummy function 94, is punched in the first field on a card.

The end directive (dummy function 96) is punched in the first field on a card, followed by the program identification. This card has no sequence number.

Where the column recognition facility is not available, unused columns and fields must be filled with zero punchings. In this case, the programmer must specify when empty columns in an alphanumeric constant (dummy function 91) are to be punched 'No Character' rather than zero.

Punched Paper Tape

All instructions, including instructions with dummy functions 92-96 are punched in the same manner. A 'Number End' is given after each element of each instruction, i.e. at each vertical line on the coding sheet after the function number. For example

```
    94                      7
0   19   B   83             2   7
1    8   B   81   B   83
```

would be punched (* indicates Number End, (indicates Letter Shift, and) indicates Figure Shift)

```
94   *   *   *   *   *   7   *   *   *
19   *  (B)  *  83   *   *   *   2   *   7   *   *
 8   *  (B)  *  81   *  (B)  *  83   *   *   *   *
```

For constants (dummy functions 90 and 91), all the elements to the right of the function are treated as a single unit, followed by a Number End. For example the constants

```
90                              1   072   53   3
91                                  CRE   D1   T
```

would be punched (* = Number End, (= Letters,) = Figures)

```
90   *   1072533   *
91   *   (CREDIT)  *
```

A run-out of 6 - 8 inches of blank tape should be provided between each stage.

The program identification should be punched as a complete dummy instruction following the end directive (dummy function 96).

5.5. How the Synthesis Program Works

Dealing with each instruction

Let us consider the task of the Synthesis Program in dealing with a very simple instruction (one not using relative addresses). The instruction is written as:-

	2		11		7	6			

and punched into paper tape or cards in the manner already discussed. The Synthesis Program is required to take this instruction and re-arrange it into the only form in which the computer can correctly execute it when the program, of which this instruction is part, is controlling the computer, i.e.:-

D35 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 1 0 0 0 1 0 0 0 0 D0

To achieve this the elements of the instruction are called separately from the buffer into the core store so that we have:-

D3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0

Reg.20 Function

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1

Reg.21"a"address

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1

Reg.22"b"address

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0

Reg.23 "C1" address

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Reg.24 "C2" address

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Reg.25 mod.

Now by shifting the contents of each register and adding, the instruction can be brought to one register in correct element sequence. The following sequence illustrates this step by step:-

0	2		24	19	4	
1	7		23	19		

D35 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 R.19 (C2, C1, 0)

2	2		19	19	10	
3	7		22	19		

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 1 R.19 (C2, C1, b)

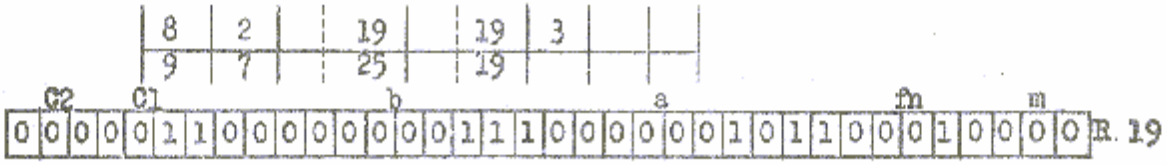
4	2		19	19	10	
5	7		21	19		

0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 1 R.19

6	2		19	19	5	
7	7		20	19		

C2 C1 b a fn

0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 1 0 0 0 1 0 R.19



5.6. Converting from Relative to Absolute Addresses

Section References

With the simple example just shown the instruction specified absolute addresses i.e. no letter was written in either the "a" or "b" address indicator columns. Where relative addresses are used (and this is the normal case) we have already seen that it is necessary for the Synthesis Program to know the address of the starting point of each section so that this address can be added to the relative address specified.

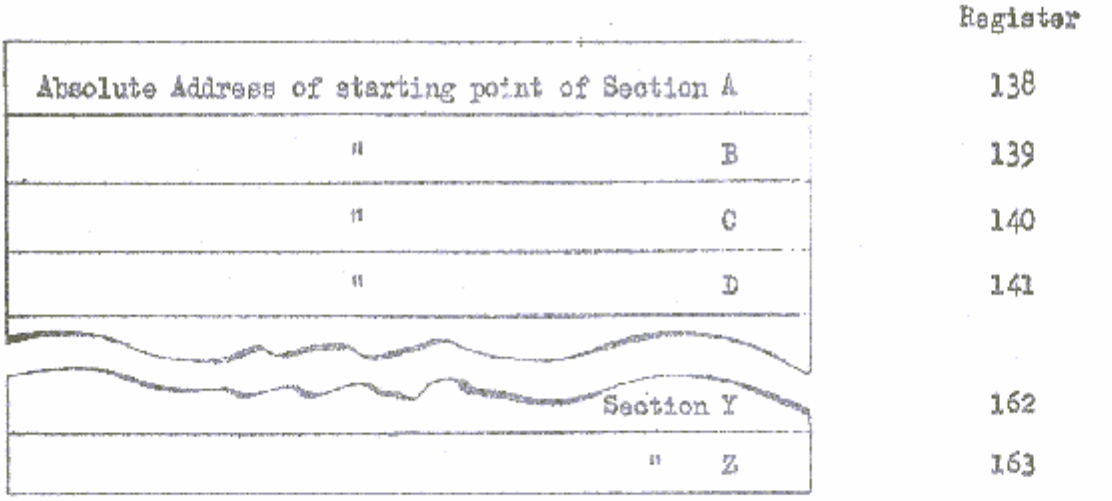
Before any instruction of a new program can be dealt with by the Synthesis Program these Section References must be read and stored in the computer so that they can be selected as required and added to the relative addresses. The first information to appear on the punched tape or cards is therefore a list of the Section References. For convenience each address is written as an instruction but to identify its real meaning to the computer a "Dummy Function" is written in the function column. The dummy function "92" indicates to the Synthesis Program that the address which follows is a Section Address and not a normal program instruction:-

	92	A	70				
	92	B	100				
	92	C	200				
	92			D	1024		

It should be noted that the core store addresses are written in the 'a' address and that drum addresses are written in the 'b' address columns on the coding sheet. This is to provide a visual distinction only, between sections stored in the core store or on the drum.

Twenty six locations are reserved by the Synthesis Instructions for the storage of Section Addresses, one for each letter of the alphabet. Four letters of the alphabet are not used as Section Indicators however, so that only twenty two of the locations are used. The letters R & S are reserved for a purpose to be discussed later, and O and I are avoided because of the possibility of confusing them with figures.

Having identified the dummy function '92' the Synthesis Program stores the address which follows in the appropriate one of a series of registers allocated to hold the Section References:-



The selection of the appropriate register in which to store each Section Reference is achieved by using the alphabetic address indicator as a modifier.

If by mistake, a Section Reference is not included at the head of the paper tape or cards containing a new program any instructions which follow with relative addresses referring to the section will have 0 added to the relative addresses.

This index remains set-up in registers 138 to 163 throughout the reading of the program.

For each instruction read before carrying out the series of instructions required to form the complete instruction in register 19, a further short sequence of the Synthesis Program is entered to add to register 21 ("a" address element) the address contained in that part of the Section Reference index specified by the "a" address indicator. Thus, if the "a" address indicator is "D" the address in register 141 will be added to the relative address in register 21.

The same procedure is carried out to modify the "b" address element situated in register 22.

Program Stages

It has already been stated that programs are sub-divided into Stages and that relative addresses are used when referring to other instructions, whether they be in the same or other Stages.

Each Stage, it will be recalled, cannot exceed 64 instructions for to do so would mean that more than one "15" instruction would be required to call the stage from the drum into the core store. This is undesirable in

organising the program and as a stage need seldon approach 64 instructions no difficulty is introduced by imposing this limit. For the same reason a stage must appear in its entirety on one track of the drum so that only one instruction is required to transfer it into the core store.

Where stages are sufficiently small, two or more stages may be stored on one track.

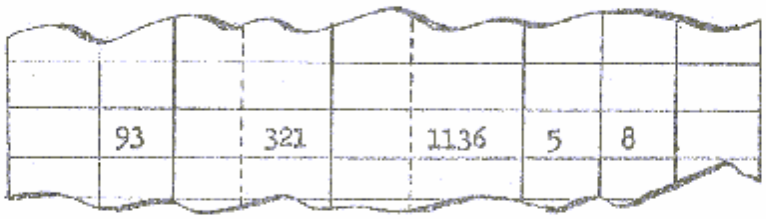
As with sections of data the programmer need not decide where to situate the program stages, either on the drum or when called into the core store, until such time as all stages have been written. Then the following information must be specified for each stage:-

(i) Address of the block on the drum in which the first instruction of the Stage is to be located

(ii) The number of blocks on the drum required for the complete Stage

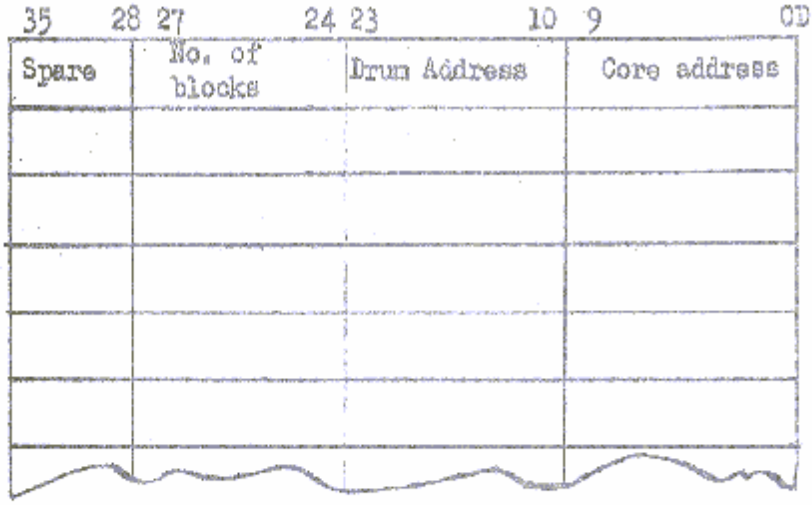
(iii) The address of the register into which the first instruction is to be called.

This information is written for each Stage as an instruction but the dummy function "93" is used to identify it to the Synthesis Program. If it is decided that Stage No 5 is to be stored on the drum starting at block 1136, it includes 32 instructions (8 blocks) and will be called into registers 321 to 352 then this would be written as:-



and the information for all Stages (in Stage number sequence to facilitate visual reference) is punched into the tape or cards immediately following the Section References and thus preceding the program stages.

Upon identification of the "93" dummy function the Synthesis Program causes the information for the Stage to be set up in a Stage Index similar to that shown for Section References. The index appears in the core store in this way:-



334 stages are allowed for in the index.

Use of the Stage Index

The Stage Index is used by the Synthesis Program for two main purposes:-

(a) When modifying relative addresses of instructions referring to other instructions e.g. | | 11 | | 0 | R | 415 | | | | |

(b) When storing the program on the drum.

Having already considered the way the Synthesis Program converts from relative to absolute addresses using the index of Section References, the use of the Stage Index for purpose (a) will be readily appreciated.

With the example | | 11 | | 0 | R | 415 | | | | | the "b" address refers to Stage 4 instruction 15. The Synthesis Program separates the Stage and Instruction numbers and then uses the Stage number as a modifier to select the information relating to that Stage from the Stage Index. The Core Store Address (D0 to D9) is then isolated from this information and added to the instruction number. In this example, if the programmer decides that Stage 4 will be called from the drum into the core store at register 620 it will be this address which will appear in the D0 to D9 positions. The Synthesis Program adds 620 to the instruction number (15) within the Stage and arrives at the absolute address (635) of the register in which instruction 15 of Stage 4 will be situated.

For purpose (b) the remainder of the information in the Stage Index for each Stage is required (D10 to D27). In fact, the Synthesis Program, having modified and assembled each instruction separately stores these instructions sequentially in a series of up to 64 registers reserved for the

purpose, until all the instructions of one stage have been dealt with. The Stage Index is then referred to by the Synthesis Program to ascertain where, on the drum, the complete Stage is to be stored and the transfer from core store to the drum is then made.

To make this process completely flexible each stage is preceded, on the paper tape or cards, by a marker which will now be explained.

Stage Markers

At the head of each stage the programmer writes a dummy instruction to indicate to the Synthesis Program that the instruction which follows is the first of a new Stage - and consequently that the previous Stage, which was being built up in the core store, can now be transferred to the drum.

This dummy instruction is written:-

| | | 94 | | | | | 16 | | |

"94" is always the dummy function used, and the Stage Number is entered in the C_1 address (in this case Stage No.16).

A new program sheet is used by the programmer when starting a new stage and the sheets are ruled to allow for the entry of the Stage Marker immediately preceding instruction 0. Where a Stage exceeds 30 instructions (the capacity of one program sheet) the Stage Marker must not be written on the continuation sheets.

On recognition of the Stage Marker, apart from transferring the previous Stage to the appropriate locations on the drum, the Synthesis Program also selects, from the Stage Index, the information relating to the Stage with which it is about to deal. This information is stored in the first register (register 165) of the Stage Index and will remain there only until the stage has been built up and transferred to the drum, at which point it will be overwritten by the information for the following Stage.

Now, with the instruction | 11 | | 0 | R | 415 | | | | it was explained how the Synthesis Program separated the Stage and instruction numbers. However, if the instruction had been | 11 | | 0 | R | 15 | | | | the result of separating would have indicated Stage 0, instruction 15, and the Synthesis Program would refer to register 0 of the Stage Index (register 165) to obtain the information relating to the Stage. As the relative address in the second example means "instruction 15 of the current Stage" and as the details of the current Stage are placed in register 165 by the

Synthesis Program, it will be seen that this relative address will be correctly converted to absolute form.

Calling Program Stages from the Drum.

When a new program is being written the programmer must decide which stages are required in the core store at any one time. It is known that to call a Stage from the drum, one "15" instruction is always required, but, although the programmer will know while writing the program which Stage or Stages are to be transferred to the core store, and the sequence in which they are to be transferred, he will not know until the program is complete the drum and core store addresses of the Stages.

This introduces the last of the relative address techniques employed in programming the EMIDEC 1101. It is simply necessary for the programmer to write a "15" instruction to call each Stage from the drum when required and in the "b" address of the instruction to enter the Stage number with the letter "S" in the "b" address indicator column. Thus, to call Stage 24 from the drum, the following instruction is written:-

		15			S	24			

The letter S is reserved exclusively for this purpose.

The Stage Index provides the Synthesis Program with all the information required to complete the instruction by adding the absolute core and drum addresses of the Stage together with the number of blocks comprising the Stage. The Stage number is used as a modifier to select the appropriate location in the Stage Index which holds the Stage details. If Stage 24 is to be stored on the drum starting at block 1136, consisting of 8 blocks (32 instructions) and is to be called into the core store starting at register 321, all this information will be found in the Stage Index, as we have already seen.

The instruction formed by the Synthesis Program, in this example, would be:-

35 - 32	31 - 28	27 - 18	17 - 8	7 - 3	2 - 0	D
8	0	1136	321	15	0	

5.7. Program Constants

Program Constants can be considered under three general headings:- Numerical, Alphanumerical, Modification. In fact, as will be shown, the distinction between the first and third of these classes is not always clear.

Numerical Constants

Examples of these are:-

(i) For setting up counters e.g. 200

(ii) As divisors for conversion from lb to tons etc. e.g. 2240

(iii) For collating out part of a word e.g. D0 - D17 = 262,143

Alphanumerical Constants

These are constants normally required for printing and on which no arithmetic operations are performed. Examples are:-

ISSUES

SPOOK

JOB 6

EXPORT

etc.

Modification Constants

These constants can really be considered as numerical constants but are classified separately, because for programming facility they are written in a different form.

Examples of these constants are already provided as source constants in registers 11 to 15 and, for instance, register 13 contains simply a "1" in the "b" address or a "D18". This could be considered as the numerical constant 262, 144 but written in this form its normal use - as an increment in a repetitive sequence - is far less apparent than if it were expressed as an instruction, thus:-

	0		0		1		

If, in a repetitive sequence, both the "a" and "b" addresses of an instruction were to be increased by 10 at the end of each cycle, the programmer writes the required constant as:-

	0		10		10		

which is exactly the way in which he thinks of the constant rather than the numerical equivalent 2,624,000.

In some cases it is also simpler for the programmer to express collating constants in this way. Should the binary digits D8 to D17 be required as a constant i.e. a binary "1" in each of the "a" address positions of a word, the constant could be written:-

		0		1023				

How the Synthesis Program deals with Constants

We will deal firstly with Modification Constants. As these constants are written as instructions they need not be specially identified to the Synthesis Program which treats them in precisely the manner outlined for assembling a program instruction. The fact that, with the examples shown these constants will appear in the computer as "STOP" instructions is entirely irrelevant, for they will never be selected and interpreted as instructions but only used by the program as numbers.

Numerical and Alphanumerical constants do, however, require to be identified to the Synthesis Program so that they can be interpreted and converted appropriately. This identification is achieved by the use of two further "dummy functions", 90 identifying a numerical constant and "91" an alphanumerical constant, e.g.

	90			467	13	2	
	90			-	20	0	
	91			M/C.ST	OC	K	
	91			ISS	UE	S	
	0		10	10			} No identification required
	0			100			

It is convenient for constants to be assembled together at the end of each stage of program and they must be immediately preceded by an "unconditional test instruction", e.g. | | 11 | | 0 | R | 500 | | | | (or in rare instances by a "Stop" instruction) to ensure that they are never treated by the computer as instructions.

Alphamuneric constants exceeding the capacity of one word (6 characters) must be broken down into two separate parts and written as two constants, each identified by the dummy function 91:-

SHORTAGE would be written

	91				N/CN/CN/C	N/C S	H
	91				CRT	AG	E

If an alphanumeric constant is less than 6 characters the more significant character positions will contain zero unless "no character" is written as above. This does not apply to paper tape where "no character" will be put in automatically.

It should also be noted that where negative numerical constants are required, as in the example shown earlier, a minus sign (-) is written immediately preceding each constant. The Synthesis Program converts signed constants into binary and then complements the binary numbers.

5.8. Function 95. Amendments to Programs on Paper Tape

It is desirable when amending a program on paper tape, to produce a new corrected tape embodying the changes which are to be made.

This rigorous approach may cause difficulties, especially in trial stages when successive alterations are called for in a long program, because a great deal of the correct tape will have to be reproduced so that amendments can be inserted in the middle.

Further, the copying process is carried out on a machine which, though in the main very dependable, can occasionally introduce errors into a previously correct portion of the tape.

Hence Paper Tape Synthesis incorporates the use of a dummy function for the amendment of a program already synthesised. This is function 95.

Corrections to the program will be fed to the computer after the original program but before the end directive (Function 96). Each correction will be preceded by a dummy instruction with function 95, which will contain in the "A" address the stage and instruction number at which the amendment commences, thus:

95	R	1014					

(In the usual convention, this implies stage 10 instruction 14)

The correction itself may consist of any number of consecutive instructions written and punched in the usual way. The new sequence will be stored in the place of the previous contents of the specified locations.

As with normal stages, the end of the correction is marked by the 94 or 95 which signals the beginning of another stage, or of another

correction; or by the end directive. (See 5.9.)

It will be appreciated that for the correct functioning of this procedure the index of stages and sections is required. It follows that if the uncorrected program is already set up on the drum, it is possible to take in the synthesis program and read only the corrections, provided that they are preceded by the index and followed by the end directive.

5.9. The End Directive

When feeding data on paper tape or cards to the computer it is necessary to provide an indication to the computer that the end of the data has been reached. In some commercial applications this might be effected by punching a dummy item no. or customer no. after the last item of data - a number such as 99999 which is outside the range of code numbers used in the application and which can therefore be recognised by the program as the "end sign".

In the same way the Synthesis Program requires to know when the last instruction of a program has been read from paper tape or cards, and the indication is provided by a further (and the last) dummy function. This is function '96' and is known as the "End Directive".

On reading this dummy function the Synthesis Program transfers the last stage of the program it has been reading to its appropriate location on the drum. At this point the complete new program has been synthesised and stored on the drum and is now available to take over the control of the computer from the Synthesis Program. This transfer of control is effected by the Synthesis Program which now calls from the drum into the core store the first stage of the new program to be executed. Having done this, the Synthesis Program forms, and carries out, an instruction transferring control to the first instruction of the Stage. Now although it is still situated in the core store, the Synthesis Program has completed its task and the program it has caused to be read is in command and Stages of the new program will be called into the core store as required. These stages will overwrite the Synthesis Program, or perhaps, cause the registers occupied by the Synthesis Program to be cleared where the registers are required for data storage.

The choice of the first stage of the new program to be called into the core store by the Synthesis Program is given to the programmer. The stage need not be either the first or last to be read. When writing the

dummy function 96 at the end of the program, the stage number of the first stage to be executed is entered in the "C_1" address position on the program sheet.

For a purpose to be explained later (5.12. RECORDING PROGRAM) the "end directive" also includes information in the 'a' and 'b' address positions. These two items of information define the area of the drum allocated to the storage of program as follows:-

'a' address = first block no. on drum in which program instructions are stored.

'b' address = last " " " " "

In the very next instruction following the end directive the programmer must write the identity number to be allotted to the program; this is explained in detail under section 5.12, Recording Program.

The complete end directive is written:-

		1st BLK	LAST BLK	1st STAGE		
	96	1024 7	1663	5		End Directive Program No.

Note:-

The program number is treated as an alphanumeric number and use may be made by the programmer of the indicator column in the program number field so that alphanumeric numbers of up to 6 characters may be used.

5.10. Preparation and Storage of Programs

Assembling and Punching a New Program

All the information required for a new program has now been illustrated, and it is now shown how the material must be prepared for punching into either paper tape or cards. Irrespective of the input medium to be used, at this point a check should be made to ensure the following:-

(i) That the program is preceded by a coding sheet listing, under dummy function 92, all the Store Section Addresses.

(ii) That the list of Store Section Addresses is followed by a list of Stage Indices (dummy function 93).

(iii) That the program is complete, and that the first coding sheet of each new stage is headed by the Stage Indicator (dummy function 94).

(iv) That an End Directive is quoted at the end of the program (dummy function 96), including the Program Identity Number.

5.11. Recording Programs onto Magnetic Tape

The manner in which a new program is synthesised has now been demonstrated and mention has already been made of the way the Synthesis Program enables the first stage of the new program to be called down and entered on recognition of the 'end directive'.

Having synthesised a program, the programmer has a choice. He may enter the program, or he may enter the Recording Program, which will store the synthesised program on magnetic tape, with the facility for recall and re-entry.

With the "End Directive" (function 96) the block numbers of the first and last program blocks on the drum were quoted. With the aid of these two items of information the Recording Program is enabled to compile instructions to read the program from the drum. In addition the Synthesis Program prepares a check total for each program block (to ensure that each time the program is read, the information stored away is correct), and these check totals together with the program itself are stored on magnetic tape. It will be remembered that the end directive was followed by the Program Identity Number, to be allocated to the recorded program for use when the program is called back from the Program Tape.

As each program is individually identified it is possible to store more than one program on a program tape, in fact the recording program provides for the storage of up to 60 programs on one Program Tape. Each time a new program is added onto the tape, or an old program is replaced on the tape, a new Program Tape is written. The Recording Program is thus a simple updating routine.

The layout of a Program Tape will be found in CP.5259. When a synthesised program is to be recorded onto a Program Tape the operator will carry out the operating instructions also listed there. This will cause the existing Program Tape to be re-recorded onto a spare tape until an end-marker is detected, at this point the newly synthesised program will be recorded onto the spare tape and finally, the end-marker will be recorded.

5.12. Replacing Recorded Programs

It is possible to record a newly Synthesised Program as a replacement to a program already on the program tape. If the identity number of the program to be replaced is punched into the B address of the instruction following the end-directive, the Recording Program will automatically replace this program by the newly synthesised program, e.g:-

	96		1024 7		1663 6	5	

End Directive
Identity number

This means, replace program 6 on the program tape by the program to be found in blocks 1024-1663 inclusive, on the drum and allocate the number 7 to the new program.

5.13 How Synthesis is Stored

Throughout this chapter we have assumed that the Synthesis and Recording program is in the computer. In fact the Synthesis and Recording program is Program No. 1 on the Program tape and may be called into the computer at any time by carrying out the Program Selection Routine. The fact that at some point the Recording Program may at some point remake the Program tape is not significant.

The Program Tape comes into existence initially by means of a one-time program which will prepare the Program Tape in the correct format, with Synthesis and Recording Program as Program 1, and other service routines as follows:-

Program No. 2. Store Print-out.
" " 3. Tape Print-out.
" " 4. Trace
" " 5. Stopping Routine.

This tape is given to all EMIDEC users with their machine and by using this tape a file of programs may be built up. Since the first five programs are already on the tape other programs should not be numbered 1 to 5.

5.14. Program Selection Routine

The first item on a Program Tape is the Program Selection Routine. When the operator requires to call in a program from the Program Tape he will carry out the operating instructions to call in and carry out the Program Selection Routine.

Briefly, the program selection routine will read the program identity number (set up in alpha), from register 16 and will read through the program tape until a marker block is found with the same identity number in word 3. The routine will then use information stored in the marker block, to read and store the selected program on the drum, and finally to call and enter the preliminary stage (of the selected program). There should always be a halt instruction in the first register of the preliminary stage, otherwise the program will be carried out immediately. The halt will give the operator time to set up peripheral equipment for the selected program. Throughout the reading operation, the program selection routine checks that the information read in is correct, by means of check totals written originally by the recording program.

5.15. Service Programs

The five service programs are those listed under 5.13. and details of these programs will be found in CP.5259.

Intelligent use of the service programs, including Synthesis, will minimize the difficulties involved in program testing. When a program is to be synthesised for the first time the following operations should be carried out:-

1. Call all service programs from the Program Tape and store on the drum.

N.B. Providing that the programmer does not use registers 17-20 in the core store, or the tracks occupied by the Service Programs on the drum, then any service program may be called into the store at any time and carried out without corrupting the program being tested.

2. Call in Synthesis and Recording Program from the Program Tape.

3. Synthesise the Program - on cards or paper tape - ensuring that full use is made of the facilities provided by the synthesis program which are:-

   3.1. A print-out of the program may be obtained at the same time as synthesis is being carried out.

   3.2. Cards may be checked for sequence.

   3.3. Cards or paper tape may be re-fed and a comparison made with what was read on the first run, any differences being printed out.

4. Record the program onto the Program Tape immediately before attempting to prove the program, this ensures that cards or paper tape are fed not more than twice, magnetic tape being the quicker and more reliable medium.

5. Having recalled the program from the Program Tape testing can begin. During testing errors may be found, in which case the program can be amended on the Program Tape as follows:-

5.1. Call in the program to be amended from the Program Tape.

5.2. Synthesise the stage and section index of the program together with the amended stages, and a new end-directive indicating that a program on the Program Tape is to be replaced.

5.3. Produce the new Program Tape.

Note

1. With paper tape it is not necessary to feed the complete version of the amended stages as in 5.2. above; instead use may be made of function 95 (sec. 5.8).

2. Full operating instructions for this procedure will be found in CP.5259.

3. It is desirable that 2 copies of the Program Tape should be kept. Operating instructions for copying the Program Tape are in CP.5259.

5.16. Synthesis and Recording Program - Amendment

Section 5 of the programming manual is now slightly out of date. The Synthesis and Recording Programs have been re-written and full details including flow-charts and coding are to be found in CP.5259 "Service Routines for Input and Testing of Programs". A general description of the new programs may also be found in CP.5265.

The following is a brief summary of the changes made:-

1. Synthesis

The program has been extended and now includes the following additional facilities:-

(a) A printed copy of a program in absolute form may be produced simultaneously with Synthesis.

(b) A comparison run may be made to check that no corruption has occurred.

(c) A check is made that no stage of program exceeds the length specified in the index.

(d) A check may be made that cards are read in the correct sequence.

(e) The program identification is read and used when recording the program on tape.

2. Recording Program

The old system of recording has been revised to allow up to 60 programs to be accommodated on one tape. The first 5 programs on any tape will be service programs as follows:-

PROG. NO.	1	Synthesis and Recording Programs
" "	2	Store Print-Out
" "	3	Magnetic Tape Print-Out
" "	4	Tracing Routine
" "	5	Stepping Routine

The remaining programs will be the working programs. Any program may be called from the tape by setting the appropriate Program Identity Number in Reg. 16, and carrying out the operating instructions for the Program Selection Routine (see CP.5259 or CP.5265).