A.T. Gough.

902 SAP

MC2/144 PROGRAMMING MANUAL

Part 2: PROGRAMMING LANGUAGES

Section 2: S.A.P. :SYMBOLIC ASSEMBLY PROGRAM

CONTENTS

Errata

In the first issue of the S.A.P. assembler, the following restrictions should be noted. These restrictions will be lifted in later versions.

(1) Page-Relative address Forms (2.1.2) are not permitted.

(2) Shifts (see 4.8.5) must be written in the form 14;+n or 14;-n.

(3) After an error (except E0 amd E16) the assembler will stop. Continuation is not possible.

(4) When the tape of 902 S.A.P. is being loaded, it will stop several yards short of the end. The initial instructions key MUST be used again to load the last section of tape.

Chapter 1: INTRODUCTION

1.1 General

The 102C/902 Symbolic Assembler Programme (S. A. P.) enables programmes to be written in a modified form of machine code which has two main advantages:

(i) Store locations may be referred to by name rather than absolute addresses.

(ii) It is possible to write instructions using constants without specifying where the constant is stored.

Programmes written in S. A. P. code are assembled using a two-pass system whereby the source tape is loaded into the computer twice, and on the second pass a binary tape of the programme is produced with a parity and sum-checking loader at the head. This tape can be entered into the computer by means of the initial instructions.

1.2 Elements

The following elements are permitted in a S. A. P. programme, and must be spaced from each other by at least one separator:

Words

Labels

Directives, including patches

Global Indentifier Lists

Comments

Trigger

1.3 Separators

Permissable separators are:

Space

Tab

Newline

There is complete page layout freedom except that there must be no more than 95 characters on one line. However the separator 'Newline' or 'Linefeed' is not permitted inside an element other than a comment or global identifier list.

1.4 Six bit Internal Code

S. A. P. operates internally in a 6-bit code, which includes the following characters, all of which are common to 920 and 903 Telecodes:-

Letters A to Z

Digits 0 to 9

Layout characters

'Tab' 'Space' 'Newline'

Printing characters

, . ; : + - * / =

( ) [ ] % & ≠

Stopcode, (i. e. Halt)

On input: a-z are stored as A-Z

Tab (920 Flexowriter)	Are stored as 'Tab'
Horiz. Tab (903 Flexowriter	
Newline (Flexowriter)	Are stored as 'Newline'
Linefeed (903 Teletype)	
Blank	Are ignored
Erase	
Carr. Ret. (903 Teletype)	

Most characters not listed above are stored as "impermissible" and give rise to error indications.

On output:

If in 903 code: 'Tab' is punched as 'space'

'Newline' is preceded by Carr. Return.

Impermissibles are punched as 'space'.

1.5 Punching Rules

A programme may be punched on one or more tapes. Each tape must end with at least one separator and stopcode; tapes may be in I. S. O. code (7 track plus parity), 903 (4100) telecode or 920 (503) telecode.

Blanks, erases and Carr. Ret's (903 Teletype) will be ignored and, apart from these characters, the first character of a tape must be Newline or Linefeed (903 Teletype). When punching programs on I. S. O. code equipment the symbol \ (reverse slash) must be used in place of £.

Chapter 2: ADDRESS FORMS

Throughout the programme the programmer may refer to a store location by any of the following forms:

(a) Absolute

(b) Identifier

(c) Relative to identifier

as described below.

2.1 Absolute

There are two forms of writing a known absolute address:-

2.1.1 Zone-relative

Form A ; B

Where A and B are unsigned decimal integers, and:

$0 \leqslant A \leqslant 4095 \qquad 0 \leqslant B \leqslant 7$

This specifies address A + 4096B

Example: 20; 1 refers to the location with decimal address 4116 (octal address 10024)

2.1.2 Page Relative

Form C * D

Where $0 \leqslant C \leqslant 127$ and $0 \leqslant D \leqslant 255$.

Thus specifies address C + 128D

Example 20 * 1 represents the decimal address 148 (octal address 00224).

2.2. Identifier

A name invented by the programmer consisting of up to 5 letters or numbers commencing with a letter. Such a name is called an Identifier. (More than 5 characters are permitted, but they will be ignored).

An identifier may be located in two ways:-

(a) The identifier may be used as a label, by inserting the identifier (followed by a separator) at any point in the program. Note that instructions and data (e.g. constants, work space locations and skips) may be labelled, and the Assembler does not distinguish between instruction labels and data names. The identifier is then associated with the address of the location into which the next word would be assembled. Note also that more than one identifier can label the same location, and that the identifier is located whether or not it is actually followed by a word.

(b) Alternatively an identifier may be located by writing it, followed immediately by = and any located available address form.

e.g. JIM=0;6

FRED=JIM+1

The identifier is then associated with the address written to the right of the equals sign. Note that this does not mean 'make the content of JIM become equal to 0;6".

2.3 Relative to Identifier

Any identifier followed by a signed integer in the range ± 2047. This will NOT be interpreted MODULO 4096, i.e. if identifier FRED labels location 4090;6, then FRED + 10 is 4;7 not 4;6.

Chapter 3: BLOCKS

A S. A. P. programme will consist of one or more blocks.

3. 1 Global Identifier List

Identifiers will be classed as Global if they are to be used in two or more blocks. Each block should start with a list of global identifiers used in that block.

The global identifier list must be enclosed in square brackets [and] and each identifier must be separated from the rest by at least one separator.

Example

[START ERROR W1 VELOC]

A global identifier must be located (see 2. 2.) once and once only in one of the blocks in which it is global. Unlocated global identifiers will be indicated by an error message at the end of the last block of the program.

3. 2 Local Identifiers

An identifier which is not included in the global list of the block in which it appears is termed a local identifier. To avoid confusion the trained programmer should avoid using that identifier in any other block. However it is perfectly legal for an identifier to be used locally in several blocks, it will have a different meaning in each.

The same identifier may be used globally and locally provided that it does not appear in the global identifier list of any block using it locally.

An identifier is said to be available at any point in a programme if it appears in the global list for the current block or is local to the current block.

A local identifier must be located (see 2. 2) once and

once only in the current block. Unlocated local identifiers will be indicated at the end of a block.

The end of a block is indicated by the global list of the next block, or by a 'trigger'.

Chapter 4: WORDS

Words are the basic elements of a S. A. P. programme. After assembly each S. A. P. word occupies one store location in the computer. Words may be written in several forms, i. e.

Integers

Octals

Functions

Special 1

Special 2

Special 3

Special 4

Instructions

All these forms are used to set a pattern of 12 bits in a computer word. The different forms are provided for the convenience of the programmer, for flexibility in writing and altering programs. In the following descriptions the convention Wd [n] represents bit n of the word to be formed. (Wd [1] represents the least significant and Wd [12] the most significant binary bit). AF, AF1, AF2 represent any of the address forms described in 2. 1 to 2. 3. AF[n] represents bit n of the 15 bit address associated with AF.

4. 1 Integers

In the range + 2047 to - 2047.

Examples +10 -200 +0

(If the value -2048 is required it must be punched in octal form i. e. &4000).

4. 2 Octals

'&' followed by up to 4 digits in the range 0-7. Note that, for example, '&6' is taken to mean '&0006'

Examples: &3777 &0036

4.3 Fractions

In the range + .9999 to - .9999.

Examples +.5 -.01

4.4 Special 1

It is suggested that 4.4 to 4.7 (Specials) are ommitted on a first reading of this manual.

Special 1 is written in the form "£AF" where AF represents any address form. The word is then the numerical value of the specified address, Modulo 4096, that is

Wd [1-12]: = AF [1-12]

Examples: £FRED

£FRED+20

£237;1 (The same value as +237)

£4095;0 (The same value as -1)

The address form is useful for loading the B register with the address of a word which cannot be addressed directly.

Example of use: (The section 4.8.2 on S.A.P. literals should be read before attempting to follow this example). If ARR is an array in the first zone of store (starting at 1000;9 say) then the following instructions (placed anywhere in store) will pick up the contents of ARR+10 and store them in ARR+20.

```
0  £ARR
4  0:10
6  £ARR
5  0:20
```

4.5 Special 2

Of the form "£AF/" where AF is any address form. "£AF" is formed as for special 1, then bits 1 to 7 are removed by the Assembler. This gives the address of the beginning of the page referenced by AF.

i.e Wd[1-7] := 0

Wd[8-12]:= AF [8-12]

Example £FRED/

This special is useful setting the modification register, in particular for saving space by using common literals to access different variables. See 4. 8. 2 for a description of S. A. P. literals. If address form FRED+1000 is used then the increment (+1000) is added to the address of FRED before the Assembler removes bits 1 to 7.

Example of use:

If TIME, VELOC, ACCEL are variables in zone 0, and all in the same data page (not page 0), say:

TIME=512;0

VELOC=513;0

ACCEL=514;0

then the following two sets of instructions placed anywhere in store will reference these variables, but the set on the left will generate 3 literals whereas the set on the right will use one common literal with an actual value of +512.

0 £TIME	0 £TIME/
4 0;0	4 /TIME
0 £VELOC	0 £VELOC/
4 0;0	4 /VELOC
0 £ACCEL	0 £ACCEL/
4 0;0	4 /ACCEL

See 4. 8. 2 for the meaning of 4 /TIME etc.

4.5 Special 3

Of the form "<AF1 AF2>" where AF1 & AF2 are any address forms, spaced by <u>one</u> space character only.

This is defined by

W [1-8] := AF2 [8-15]
W [9-11] := AF1 [13-15]
W [12] := 0 .

This special is useful for setting the pointer register and for sub routine entries.

Examples

< START 0;0 >
< 0;0 FDATA>
< INT IPOINT >

If the address form 2.3 is used (e.g. FRED+1000) then the increment (+1000) is added to the address of FRED before the assembler removes the appropriate bits from the address form.

Examples of use

(1) To set up a pair of words containing the 15 bit address of a label for an indirect jump or sub-routine entry, e.g.:

```
ASTART   <START 0;0>
         £START
```

Then 11 ASTART would cause program control to be transferred to the instruction labelled START, wherever that was assembled in store.

(2) To load the pointer register. The instruction sequence:

```
14  1:65
<0;0 FDATA>
```

will load the pointer register (D register) to point to the page in which FDATA is located.

(3) To set up interrupt starting data in locations 128 etc. In locations 128 and 129 set:

<INT IPOINT>

£INT

Then when interrupt occurs control will be transferred to the instruction labelled INT, and the D register set to point to the page in which IPOINT is located.

The identifiers used in specials must be "available" but need not be "located" at the point of the programme.

4.7 Special 4

Of the form "£AF = ", this may only be used as a literal in the construction of "long jumps". AF may be any address in the same 4K block as the instruction using the special. It is used to set the modification register for jumping to AF. "£AF = " will in general be a multiple (±) of 256, except where the difference between AF and the address of the jump instruction is 128 + a multiple of 256 in which case "£AF = " will be an odd multiple of 128.

See 4.8.4 for a description and example of the use of Special 4.

4.8 Instructions

May take several forms. These commence with a function in the range 0-15, followed by at least one separator and a permitted address form.
(In the forms listed in 4.8.2 below only; the function may be preceded by a "/").

The identifiers used in addresses must be "available" but need not be "located" at that point in the programme.

4.8.1 Machine Code Form (Decimal Address)

Any instruction may take the form,

F M;N where F, M and N represent decimal numbers, and:-

F is the function in the range 0-15

M is the mode, 0 or 1

N is the address in the range 0-127

F must be followed by at least one separator M, colon and N must not be separated.

i.e. Wd[1-7] := N

Wd[8] := M

Wd[9-12]:= F

Examples: 4 0;20

15 1;127

4.8.2 Store - Addressing Functions

Functions (other than relative jumps) which address the store may be written as

F A or /F A

where F is the function in decimal, i.e.:-

0, 1, 2, 3, 4, 5, 6, 10, 11, 12 or 13.

and A is one of:-

(a) An address form with a value in the range 0 to 127.

e.g. 4 10;0

(b) The form '/AF' where AF is any address form, and /AF represents the least significant 7 bits of AF (i.e. Wd[1-7] := AF[1-7]

e.g. 5 /FRED

(c) Any literal form, see below

e.g. 4 + 2

If F is preceded by / then the mode bit will be set in the instruction (i.e. Mode = 1). If F is not preceded by / then the instruction will always be assembled with Mode 0.

4.8.2.1 Literals

The S.A.P. Assembler provides a facility for making constants available in a program and allocating storage to these constants automatically. The programmer simply writes the constant into the address part of the instruction. Such constants are known as "literals" (or "S.A.P. literals"). (Note that 902 machine code does not have a literal address form, S.A.P. literals are always placed in a separate 12 bit word, and the address of this word is inserted in the instruction address (bits 1 to 7) by S.A.P.)

A literal may be written in any of the word forms listed in Chapter 4 except instructions (4.8).

Examples of literals used in instructions:-

```
4  +2
2  -1000
6  &0777
12 +.5
4  £FRED
0  £FRED/
4  <0;0 DATA>
0  £LAB=
```

Literals may only be used after functions 0, 1, 2, 4, 6, 12 and 13. This restriction gives some safeguard against misuse and accidental overwriting of literals, but it is still possible

for a program to corrupt literals by mistake.

If the function of the instruction is preceded by / then the literal will be allocated an address on the current data page. If it is not preceded by / then the literal will be placed in page 0. (See 5.6 for further details).

Example,

to load +1 into the A register the programmer could either:

(a) arrange to place +1 in some known location say 60;0 and write:

4 0:60 (Absolute address)

(b) or he could write C1 +1 in his data space and write instruction:

4 C1 (The exact location of C1 need not be known)

(c) or he could write:

4 +1

This is easier to read and understand than (a) or (b) and +1 will automatically be allocated space and shared with any other literals with value +1, & 0001, etc..

4.8.3 Short jumps (Functions 7, 8 and 9). Jumps to addresses which are within $\pm$ 127 words of the current address may be written either as:

(a) F ;+n or F ;-m

where n and m are single digit decimal numbers and F is a function digit 7, 8 or 9.

Examples: 10 Y

10 X

7 ; +2 (Jump forward 2 if A=0)

8 ; -3 (Jump back to 10 Y)

(b) F AF

where AF is any address form representing an address within ± 127 of the current address. The mode bit for a jump back is inserted automatically (note that /F AF is illegal in this case).

Examples 8 LAB

7 START +1

4.8.4 Modified Jumps ("Long Jumps")

Functions 7, 8 or 9 may be used to transfer control to any address in the current zone (i.e. the block of 4096 words in which the jump instruction is placed. The special 4 provides a convenient means of writing jumps which are likely to be more than 127 words long. To jump to any address form AF (normally a label) write:-

0 £AF=	0 £AF=	0 £AF=
8 =AF	7 =AF	9 =AF

Examples:

0 £LAB=	0 £START+1=	0 £END=
8 =LAB	7 =START+1	9 =END

The mode bit for a jump backwards is inserted automatically. The jump instruction 7, 8 or 9 must not be preceded by /. However the literal £AF= may be placed in the current data page by writing

/0 £AF=

4.8.5. Shifts

Shifts may be conveniently written in the form

14 ↑+n or 14 ↑-n

where n is an integer. The range of this instruction form is from 14 ↑-32 though 14 ↑+0 to 14 ↑+31 This instruction form implies "multiply by 2↑±n"

or {	shift left n places if positive sign.	14	↑+n
	shift right n places if negative sign.	14	↑-n

Chapter 5: DIRECTIVES

5.1 Start

The first element of a programme must be the directive "* START" followed by a string specifying the hardware in use.

Examples:-

*START;22

*START;21

The first and second digits are $\frac{1}{4096}$ x the store size of the COMPILE & RUN computers respectively, and this may be in the range 1-8. Thus in the 2nd example above, S. A. P. is being used on a 8192 word computer, to compile a programme which will run on a 4096 word computer.

Separators are not permitted within this directive.

5.2 Skip

The directive ">N where N is an unsigned iteger, called a skip will cause N locations to be left undefined (for workspace). ($N \leqslant 4095$).

Example: >20

5.3 Programme Pointer

"* PROG" Locate words from the address held in an Assembler variable known as "Progptr" onwards, incrementing "Progptr" by one after each word, and by the appropriate amount after each skip.

There are two forms of *PROG directive

(a) *PROG

causes words to be located from the point previously reached under the last *PROG directive. If *PROG has not been used "Progptr" will have value 256;0.

(b) *PROG = AF

where AF is any located available address form.

"Progptr" is then set to the value of AF

Example: *PROG=0;1

5.4 Data Pointer

Directive *DATA causes words to be located from the address held in the Assembler variable known as Dataptr, onwards. After this directive "Dataptr" is incremented by one after each word, and by the appropriate amount after each skip.

There are two forms of *DATA directive.

(a) *DATA

(b) *DATA=AF

where AF is any located available form. In (a) "Dataptr" takes its previous value (initial value undefined). In case (b) "Dataptr" becomes equal to the value of AF

Example: *DATA=DFRED

(DFRED must have been located before this directive was reached, e.g. by DFRED = 512;0).

Note carefully that the S. A. P. variable "Dataptr" has no relation to the hardware D register (Pointer register) at Assembly time or run time. However it may be important for the programmer to form a relation between the "Dataptr" value at assembly time and the value actually loaded into the D register at run time (see 5.6)

5.5 Patch

"↑AF" locate words from the address specified by AF, onwards. AF is any located available address form:

Examples: ↑560;1

↑START+20

This facility is a directive to stop placing words consecutively from the address held in "Progptr" or "Dataptr" but to place them consecutively from the address indicated by the patch. At the end of a patch sequence compilation

of the main programme can be continued by the directive * PROG or * DATA.

5.6 Location of Literals

S. A. P. will locate literals in Mode 0 instructions from 127 downwards. (i. e. in page 0).

Literals preceded by a Mode 1 instructions (e. g. /4 +1) will be located from the top of the 128-word page indicated by the current value of "Dataptr", modulo 128 (i. e. the current data page).

It is the programmers responsibility to ensure that, when compiling an instruction using a "Mode 1" literal, "Dataptr" is set to the same page that the computer pointer register (D register) will be set to when that instruction is obeyed at run-time.

This is most conveniently arranged by adapting the recommended standard program layout: Before each major section ("Chapter") of instruction code (which may be one or more S. A. P. blocks) the data which is associated with that code is declared under *DATA, limited to a single page (128 words) known as the local data page for that Chapter. The *PROG for the chapter follows the code declarations. At each entry point to the Chapter, the pointer register (D register) is loaded with the address of the local data page. As far as possible the programmer should avoid changing the D register within that Chapter. When it is essential to alter the D register it should be restored to point to the local data page immediately the operation is complete. Mode 1 literals should not be used in instructions unless they will always be obeyed with the D register pointing to the local data page.)

(If the same instruction is to be obeyed with varying pointer-register values "Mode 1" literals must be avoided and the constants required located explicitly by the programmer on <u>each</u> page that they will be required.)

Note that literals are shared whenever this is possible e. g. 4 +63 and 6 &0077 would share the same Page 0 literal.

S. A. P. will record the highest location used on each 128-word page for words, (or reserved by skips) and the lowest on each page used by literals: overflow will be indicated should these crash.

Chapter 6: COMMENTS

A comment starts with a '(' and ends with a matching ')'. All characters within the brackets are ignored and a "bracket count" will be kept so that matching internal brackets are ignored. However, only internal code characters are permitted (see 1.4) and 'Stopcode' must not be used.

Example: (THIS IS A COMMENT (X;=Y+Z;))

6.1 Titles

If the first character inside a comment is a '*' it is called a title.

Example: (* TITLE;PROGRAM A;21/10/68)

Chapter 7: TRIGGER

The end of a programme is indicated by a trigger. When a trigger is read, all global and no local identifiers are 'available'.

Triggers may be of the form % AF, where AF is any located available address form, thus local identifiers must not be used, but any identifier declared as global anywhere in the program may be used. When S. A. P. reaches the next stopcode after a trigger it will punch the trigger on the binary tape and the sumcheck for that tape. When the binary tape is read into the store under initial orders the programme will be triggered at the specified address, (provided the sum and parity checks succeed).

If no programme trigger is required the programme should end % %.

Example of trigger: %START

Chapter 8: OPERATING INSTRUCTIONS

The S. A. P. Assembler may be used to translate a number of programs into binary form, and for efficient use of the computer it is recommended that programs are assembled in batches, for running at a later stage:-

(1) Load the tape 902 SAP in the reader and press the initial instructions key down. The tape should read up to the last non-zero character and stop. If it stops elswhere or if there is output on the punch. the tape has not been read correctly, or is a faulty copy.

(2) Load the first tape of a SAP program in the reader. Set up keys 1 to 6 on the control panel (w/g) to control print-out as described below.

(3) Change key 1 1, the tape will then be read for the first pass.

(4) Input subsequent types of the same program (if any) by changing key 11.

(5) When all tapes have been read for pass 1, run-out blanks on the punch, load the first tape of the program in the reader again and change key 11.

(6) Re-read subsequent tapes of the program (if any) in the same order as for pass 1, by changing key 11.

(7) When all tapes have been read for Pass 2, run-out tape and tear off. Return to step (2) if more programs are to be assembled. If errors have occurred it may be necessary to use the re-start facility, see Chapter 11.

(8) To run the assembled program, load the binary tape produced and enter initial instructions (similar to step (1)).

On the first pass of the tapes, S.A.P. will punch information according to the word generator setting.

(If the 12th (m/s) key of the w/g is down, S.A.P. will wait when it next reads the w/g).

If key 1 is down, print-out will be in 920 Telecode, and it it is up, the 903 Telecode (suitable for Teletype or Flexowriter - i.e. "Tab" will come out as "space" and "Newline or Linefeed" will be preceded by "Car. Ret. ").

If key 2 is down, label addresses will be punched in Octal. If key 3 is down they will be punched in decimal.

If key 4 is down, local label addresses will be punched. If key 4 or 5 is down, global label addresses will be punched.

If key 4, 5, or 6 is down, titles will be copied and a store map punched on reaching a trigger.

After the 2nd pass of the tapes, a revised store map will be punched.

The store map obtained at the end of the 2nd pass will include literals containing identified addresses not located at the time of reading on the first pass; the literals themselves will not be located until they are read on the 2nd pass.

Errors detected on either pass will be punched when detected in the Telecode indicated by w/g Key 1, (irrespective of the settings of keys 2-6), and, on the 2nd pass, output of the binary tape will stop.

In the case of a RUN: store overflow on either pass, a store map will be punched.

Chapter 9: MISCELLANEOUS ERRORS

9.1 Locations Reserved for Tape Loader

As locations 0-127 will be mainly workspaces the binary tape loader occupies locations 16;0 to 49; 0 inclusive.

Any attempt to locate words in locations below the high end of the loader will be an error, although they may be reserved for workspaces, e. g. WS=20;0

9.2 Programming through a 4K block

Programme may not cross the boundary of a 4096 word block of store unless a *PROG or directive is given. Any attempt to progamme or short jump (unmodified) through a 4K block boundary will give an error indication.

9.3 Data Page Overlow

When compiling *DATA and using 'skip' to reserve several locations for workspace, page overflow (provided there are no literals on the page) will give rise to a warning. (If literals are present, an error will be given).

Chapter 10: ERROR INDICATIONS

Error No.	Meaning
0	Unlocated Identifier
1	General contextual error
2	Parity error on source tape
3	Label declared twice
4	Violation on one of the following interlocks. (a) Elements other than comments (and stopcodes) before *START directive. (b) No , *PROG, or *DATA before the first word or skip. (c) No globals list before first word. (d) Two *START directives in one programme.
5	Tapes read differently on second pass to first pass. (a) Different *START directive (b) More blocks on second pass than first (c) Label address different (d) Identifier not in dictionary on second pass
6	'Progptr' or 'Dataptr' incorrectly located
7	Address error
8	Impermissable character
9	Address form which must be located on first pass is not

10	Number outside permitted range
11	Dictionary overflow
12	More than 95 characters to a line
13	Data page full. (data and literals crash).
14	Attempt to overwrite binary loader (Loc's 16;0 to 49;0)
15	Programme spills over 4096 word block boundary
16	Address form greater than size of store permitted.
17	No linefeed or newline at start of tape
18	Warning that a skip straddles a page

After an error 0 S. A. P. will continue to read in tape to find further errors, but the punching of binary tape is inhibited on the second pass.

Error 18 will not inhibit punching of the binary tape.

Chapter 11: RESTART FACILITY

After most errors, the S.A.P. assembler stops output of binary tape on pass 2, but will continue to scan for further errors. If an error does cause a stop, the assembly of the same or another programme may be started again (after such an error, or at any time, e.g. if the wrong tape has been loaded) by setting the interrupt selection switch to 'Manual' and pressing 'Interrupt'.

S.A.P. will then wait for the first tape of the correct programme to be loaded and the 11th key of the W/G moved from 0 to 1.

Chapter 12: DUMP FACILITY AND CORRECTION COMPILATION

To enable program corrections to be compiled without recompiling the whole programme, S. A. P. provides a dump facility.

To dump S. A. P. and the DICTIONARY of any programme just compiled, move W/G key 10 from 0 to 1. (The resulting tape is a sum and parity checked binary tape of the relevant areas of store).

To compile a correction to the programme, load the dump into store* and compile the correction by making 2 passes in the usual manner.

The correction only has "access" to the global identifiers of the original programme. It MUST NOT contain a *START directive, but MUST contain a trigger (or %%). It may be on more than one tape, and contain more than one block.

---

* Of the same or another computer; but if another computer is used it must have at least as much store as declared in the *START directive of the original programme.

Chapter 13: STORE USED

The S.A.P. assembler and its workspace occupies store from 0;0 to 3600;0 approximately. The rest of the store (as specified for the COMPILE computer in the * START directive) is used to hold the dictionary and literal lists. Each dictionary item (global or local identifier) takes 4 words of store. Each literal takes one word, and four words are used for every page into which program or data is stored.

At program run time the whole of the store (specified in the * START directive for the RUN computer) is available to the assembled program, except that only workspace locations may occupy 16;0 to 49;0.