Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

Introduction

Hatfield School is situated on the same campus as the Hatfield Polytechnic, and has for several years had quite extensive use of the Polytechnic's computer facilities. In September 1970 the Polytechnic began to operate a multi-access scheme using a P.D.P 10. Before this time the computer science department had experimented, on an Elliott 803, with timesharing using a simulated hypothetical machine (the BBC machine) which allowed three users to share the same ~~assembler~~ hardware on a timesharing basis using the BBC3 assembler. The basic aim of the BBC machine was to teach the processes of a computer to pupils from various educational establishments in the area.

The sucess of BBC3 encouraged the department to create a new hypothetical BBC computer within the PDP 10 and this was called BBCX. The specification of BBCX was completed at the end of last year and a programmer from the Computer Centre began work on the 'hardware' of the machine at the beginning of this year

The software for BBCX is quite clearly divided into three sections; the monitor, assembler and compiler. These are known as the three negative pages and are shared by all users. In addition each user has complete control over three pages (1,2,3) and semi-control over page 0 which contains information pertaining to that user (eg input buffer,

Specification pages 19-21

Index Number

umber of Sheet

Name

The name need not be written on any sheet after the first

Number Question must be in this

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

assembly storage pointer). Thus each user has his own pages 0 to 3 but shares the negative pages -1 to -3. The BBCX machine uses the timesharing facilities of the PDP 10 and hence the high segment (pages -1, -2, -3) must remain unchanged. It is for this reason that in writing the software it has been necessary to store all information about a particular 'job' in the user's page 0, (the compiler is allotted cells 640-895 for this purpose) and only the accumulators and cell 8 (page cell) can be changed in any of the negative pages.

Whereas the low level assembler translates instructions on a line by line basis, the compiler produces code for the user in response to certain commands. I have been given the task of writing this compiler and this project concerns one command in the compiler. The compiler commands fall into sections; those which require very little programming (e.g. MESSAGE) and those requiring the evaluation of an arithmetic expression (ASSIGN, IF, LOOP). This project contains the entire programming for ASSIGN and this has been written in such a way that it will be easy to adapt the reverse polish and compiling routines to produce the code for IF and LOOP.

It may have been easier to have written these three commands in one go, but I felt that this would have been a rather large program to test at a time particularly as I have had no previous experience of writing in BBCX. Besides the Polytechnic has begun to

teach BBCX this summer-term and one command in its entirety in the compiler is more useful than several commands in the 'pipeline'.

Unlike many hypothetical machines most of the software (Monitor, Assembler, Compiler') has been written within the BBCX machine. Thus, this program has been written in a semi-~~compiled~~ translated version of BBCX' using a somewhat limited selection of Monitor commands and a 'dummy' Assembler which will not recognise errors and which requires a semi-translated form. This has been necessary because the hardware was slightly behind schedule, which has held up work on the three negative pages.

Computer Facilities.

The Hatfield Polytechnic has a P.D.P 10 with 96 K core and fifty simultaneous online terminals. Hatfield School has the use of one terminal. The program is written in a semi translated version of BBCX, a specification of which is enclosed.

' Page 27 of Specification

Number question must be in this margin

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

ASSIGN.

The definition of ASSIGN, the format of the arithmetic expression and the generated code were specified[1] but the method was left to me under the restriction that the entire code for the compiler must fit into one page.

The code for ASSIGN is generated using three passes. The first pass checks that the brackets are correctly nested and that there are not too many colons. The second pass turns the arithmetic expression into reverse polish notation. The third pass changes the semi-compiled instructions into the code.

Reverse Polish Notation.

In reverse polish the items in an arithmetic expression are divided into two groups; the operators and the operands:

Operators :- +, −, *, /, ↑, ⊖, (,), extra code, [,],
user function, array identifier, ⊕

Operands :- identifiers, numbers.

(NB throughout ⊖ and ⊕ denote monadic minus and plus)

All elements in the group of operators have a number associated with them, known in this project as their precedence code.

[1] See the specification of ASSIGN pages 32-34

Index Number

Number of Sheet

Name

The name need not be written on any sheet after the first

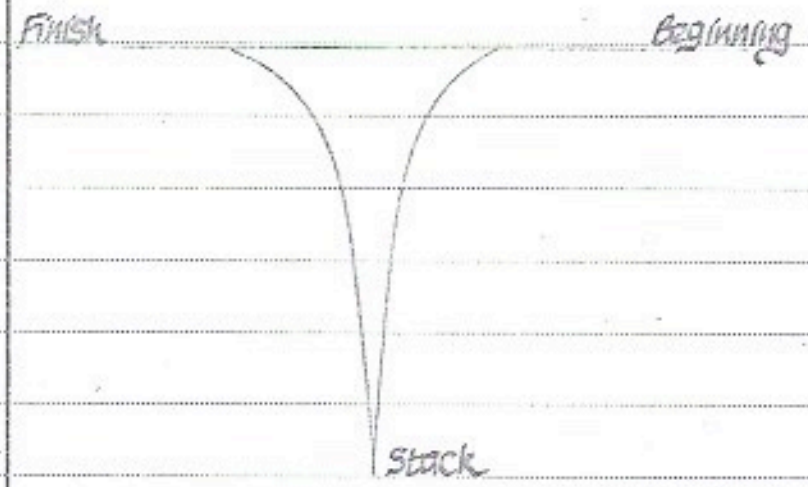
Number question must be in this

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

Precedence code	Operator (s).
0	extra code, user function, array identifier, ⊖
1	↑
2	*, /
3	+, −
4	(,), [,]

The method for turning an arithmetic expression into reverse polish notation is best described by using a diagramatic as shown below.



The arithmetic expression starts off at Beginning and each element is considered in turn. All operands go along the straight track to the Finish. Operators go down onto the stack. Here, if the precedence code of the top item is greater than or equal to the precedence code of the next to top item, this item takes the track up to the Finish. This is repeated until the precedence code of the top item is less than that of the next to top item. There is an exception to this rule; an open parenthesis (or [ ) acts as a stop on the stack and nothing can be unloaded beneath it until a closed parenthesis ( ),]) joins the stack. In this eventuality all characters are unloaded in

also items of top precedence are just added to the stack without any testing

Index Number_____

Number of Sheet_____ Name_____

The name need not be written on any sheet after the first

Number Question must be in this

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

the order first-on, last-off regardless of precedence until a matching open parenthesis is encountered; then both parentheses are completely erased, monadic plus is also simply ignored. On reaching the end of the arithmetic expression anything left on the stack is unloaded into Finish (first-on, last-off order)

An example will perhaps clarify the above explanation. Take the arithmetic expression:

A* SQRT(Σ[D*⊖B]+F)-3/E

(Σ means that X is an array identifier as opposed to an simple identifier)

A (Σ[D*⊖B]+F)-3/E

SQRT
*

'SQRT' rests on top of '*' since precedence of 'SQRT' is ø and of '*' is 2.

A [D*⊖B]+F)-3/E

Σ
(
SQRT
*

Although array identifiers and 'SQRT' both have a precedence of ø, 'SQRT' is not unloaded because of the presence of the open bracket.

Index Number _____

umber of Sheet _____ Name _____

The name need not be written on any sheet after the first

Number Question must be in this margin

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

ADB J+F)-3/E

⊖
*
(
c
(
SQRT
*

Again '(' protects 'c' from being unloaded by '*'

ADB⊖* +F)-3/E

c
(
SQRT
*

'J' has unloaded the stack down to '(' i.e. '⊖' & '*' in the order shown.

ADB⊖*cF)-3/E

c +
(
SQRT
*

'+' has unloaded 'c' since '+' has precedence 3 and 'c' precedence ø; '(' still protects 'SQRT'.

ADB⊖*cF+SQRT* 3/E

-

')' has unloaded '+' and erased '(' The following '-' has loaded 'SQRT' and '*' since these have precedence of ø & 2 whereas '-' has precedence 3.

Index Number_____

umber of Sheet_____

Name_____

The name need not be written on any sheet after the first

Number Question Y must be n in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

ADB⊖*CF+SQRT*3E

Final unloading of the stack results in the arithmetic expression written in reverse polish.

/

ie. ADB⊖*CF+SQRT*3E/–

Storage of reverse polish in the BBC compiler.

In order that the final conversion of the reverse polish into compiled code be easier, the operators are split into four groups which are distinguished by the setting of the type bits[1] in the reverse polish code of each.

P-word : the group of operators which will have to have an address added in the compiling routine; these are stored as P-words since they are used most often and there will be no need to change their type.

S-word : the group of operators which will not have to have anything added in compilation.

F-word : array variables since although these are operators only the address connected with the array is stored; this is because a full instruction would read: TAKE 6,*A(6)

[1] for the normal use of type bits see page 2 of the specification

umber of Sheet_____ Name_____

The name need not be written on any sheet after the first

Number Question if must be n in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

thus filling the index register which is reserved for the precedence code.

I-word: open parentheses, since these are not finally compiled. Closed parentheses are stored like open parentheses since these do not remain on the stack.

Operands are stored as I-words with the address connected with the identifier or constant.
':=' is stored as "&&&&" i.e. (ØØØØØØØØØ)
'newline' is stored as "&&&A" i.e. (ØØØØØØØØ1)

The generated reverse polish is stored in cells of the user's page-Ø starting at 645. The destination address for ASSIGN is stored as an I or F word, depending upon whether it is a simple identifier or an array identifier, in 644 in page-Ø. Before an operator from the stack is added to the reverse polish its precedence code is removed by a logical AND with (177707777).

The reverse polish codes for operators are given on the next page, in all cases the three bits which are normally used to specify an index register[1] are used to indicate the precedence code.

[1] page 3 of the specification

Index Number _____

Number of Sheet _____ Name _____

The name need not be written on any sheet after the first

Number Question You must be in this m

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

Table of reverse polish codes for operators.

Type grouping.	Operator as input	Storage of operator	Octal Representation Type	Function	Acc	Index Reg. (Address)	Address + fg. bit & indirect bit.
P-word	+	ADD 6,Ø	P	Ø4	6	3	ØØØØ
	−	SUBT 6,Ø	P	Ø5	6	3	ØØØØ
	*	MULT 6,Ø	P	Ø6	6	2	ØØØØ
	/	DVD 6,Ø	P	Ø7	6	2	ØØØØ
	↑	POWR 6,Ø	P	34	6	1	ØØØØ
	⊖	TNEG 6,Ø	P	12	6	Ø	ØØØØ
S-word	SQRT ⋮ ABS (Extra codes)	EXTRA 6,1	S	77	6	Ø	ØØØ1
		EXTRA 6,18	S	77	6	Ø	ØØ22
	LABEL user subroutine let address assoc. be $1000_{10} \equiv 1750_8$	JUMP 7,*1000	S	60	7	Ø	5750 (1750 + 4000 for indirect bit)
Fword	ARRAY array identifier let address assoc. be $1000_{10} \equiv 1750_8$	NIL Ø,1000	F	ØØ	Ø	Ø	175Ø
I-word	(or) [or]		I	ØØ	Ø	4	ØØØØ

umber of Sheet _____ Name _____

The name need not be written on any sheet after the first

Number Question 1' must be n in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

Example of storage of reverse polish.

Input statement to ASSIGN is :-

G[H+I] := A * SQRT(C[D * ⊖B] + F) − 3/E

G[H+I] in reverse polish = HI+G

A*SQRT(C[D*⊖B]+F)−3/E in reverse polish = ADB⊖*CF+SQRT*3E/-

(see the example worked through in the section about reverse polish, page)

In the following table the addresses associated with the necessary identifiers and constants is given below:-

	denary	octal
A	1022	1776
B	1020	1774
C	1018	1772
D	1016	1770
E	1014	1766
F	1012	1764
G	1010	1762
H	1008	1760
I	1006	1756
3	1005	1755

(NB as before X stands for an array identifier)

The following table shows how this arithmetic expression would be stored in page 0.

Number of Sheet _____ Exam Number _____

Name _____

The name need not be written on any sheet after the first

Number Question Y must be n in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

Address in page 0	Code (given in most readable form, independent of type)		Octal Representation
644	+1010	(G)	F ØØ ØØ 1762
645	+1008	(H)	I ØØ ØØ 176Ø
646	+1006	(I)	I ØØ ØØ 1756
647	ADD 6, Ø		P Ø4 6 ØØØØØ
648	"&&&&"		S ØØ ØØØØØØ
649	+1022	(A)	I ØØ ØØ 1776
650	+1016	(D)	I ØØ ØØ 177Ø
651	+1020	(B)	I ØØ ØØ 1774
652	TNEG 6, Ø		P 12 6 Ø ØØØØ
653	MULT 6, Ø		P 06 6 Ø ØØØØ
654	+1018	(C)	F ØØ ØØ 1772
655	+1012	(F)	I ØØ ØØ 1764
656	ADD 6, Ø		P Ø4 6 Ø ØØØØ
657	SQRT 6,		P 77 6 Ø ØØØ1
658	MULT 6, Ø		P Ø6 6 Ø ØØØØ
659	+1005	(3)	I ØØ Ø Ø 1755
660	+1014	(E)	I ØØ ØØ 1766
661	DVD 6, Ø		P Ø7 6 Ø ØØØØ
662	SUBT 6, Ø		P Ø5 6 Ø ØØØØ
663	"&&&A"		S ØØ ØØ ØØØ1

N.B all precedence codes are removed before storing in page 0

Number of Sheet _____ Exam Number _____

Name _____

The name need not be written on any sheet after the first

Number Question Y must be n in this in

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

Compiling the code from the generated reverse polish.

This is the province of the routine called 'COMPILE'. As can be seen from the examples of code generated by ASSIGN in the specification' the code makes extensive use of the hardware stack (see specification for details on the hardware stack - page 16). This makes the task of compilation easier since working space in the form of identifiers does not have to be found.

The routine is divided into two sections; firstly the compilation of the code on the right of the ':=' sign leaving the result as the only item on the stack and secondly the compilation of the code on the left of the ':=' if the destination is an element of an array, leaving the subscript on top of the stack. This order of compilation is necessary since the instruction:-

PUT 6, * A(6)

uses the top item of the stack as the subscript and the next to item as the result to be stored in A(6).

The general method for compiling code from the reverse polish in page 0 involves considering each item in the reverse polish in turn. If the item is an address connected with an identifier or constant (type = I) then this address (A) is used to compile the code:-

TAKE 6, A

'specification pages 32, 33, 34.

Number of Sheet _____ Name _____

The name need not be written on any sheet after the first

Number Question Y must be n in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

If the item is a function (type = P) (e.g. ADD 6,Ø) then this is used to compile the code

ADD 6,6.

There is an exception however if a function follows an address: in this case the code compiled is:-

ADD 6,A

since this is equivalent to the two instructions

TAKE 6,A

ADD 6,6.

If the item is an array identifier (type = F) then the code:-

INT 6,

is always compiled to ensure that the subscript is an integer, followed by either

TAKE 6,*A(6) or ADD 6,*A(6)

according to the same rules as normal identifiers.

If the item is an extra code or user subroutine (type = S) then the instruction is compiled as it is (with type changed)

e.g. EXTRA 6, 1 or JUMP 7,* L (where L is address connected with LABEL)

When both sides of the ':=' have been compiled in this way a final instruction is added which is either:-

PUT 6,A (if the destination is a simple identifier)

or PUT 6,*A(6) (if the destination is an array identifier)

Index Number_____

Number of Sheet_____

Name_____

The name need not be written on any sheet after the first

Number Question Y must be n in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

An example of the result of 'COMPILE' on generated reverse polish

If the 'COMPILE' routine were entered with the reverse polish in page 0 as given in the example on page , then the following is a simulation of code compilation.

The routine initially searches for the S-word zero that marks the position of the ':=' (e.g. cell 648) and a pointer is set to MOCKS 649

Cell under consideration	Type	Content	Code compiled	Remarks	new value of pointer
649	I	+1022	TAKE 6,1022	650 does not contain a function.	MOCKS 650
650	I	+1016	TAKE 6, 1016		MOCKS 651
651	I	+1020	TNEG 6, 1020	652 contains TNEG 6, so pointer points to 653	MOCKS 653
653	P	MULT 6,0	MULT 6,6	652 did not contain an identifier	MOCKS 654
654	F	+1018	EXTRA 6,12 TAKE 6,*1018(6)	INT 6,0 to make subscript an integer array identifier'	MOCKS 655
655	I	+1012	ADD 6,1012	656 contains ADD 6,0	MOCKS 657
657	S	SQRT 6,	EXTRA 6, 1	EXTRA 6,1=SQRT 6,'	MOCKS 658
658	P	MULT 6,0	MULT 6, 6	657 did not contain an identifier.	MOCKS 659
659	I	+1005	TAKE 6, 1005		MOCKS 660

'There is a list of extra codes on pages 21 to 22 of the specification.

umber of Sheet_____ Index Number_____

Name_____

The name need not be written on any sheet after the first

Number Question f must be n in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

Cell under consideration	Type	Content	Code compiled	Remarks	new value of pointer
660	I	+1014	DVD 6, 1014	661 contains DVD 6,Ø	MOCKS 662
662	P	SUBT 6, Ø	SUBT 6, 6	661 was not I or F and	MOCKS 663
663	S	"&&&A"		pointer is now changed to MOCKS 645 to complete the second phase.	MOCKS 645
645	I	+1008	TAKE 6,1008	646 is not a P-word	MOCKS 646
646	I	+1006	ADD 6, 1006	647 contains ADD 6,Ø	MOCKS 646
648	S	"&&&&"		moves pointer to MOCKS 644 to compile final instruction(s)	MOCKS 644
644	F	+1010	EXTRA 6, 12 PUT 6,*1010(6)	INT 6,Ø to make subscript integer	

Number of Sheet _____ Name _____

The name need not be written on any sheet after the first

Number Question Y must be n in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

Thus the final code produced by the assignment statement :-

ASSIGN G[H+I] := A * SQRT(C[D*-B]+F) - 3/E

is :-

```
TAKE   6, 1022      ; A
TAKE   6, 1016      ; D
TNEG   6, 1020      ; B
MULT   6, 6         ;
EXTRA  6, 12        ; INT 6,
TAKE   6, *1018(6)  ; C
ADD    6, 1012      ; F
EXTRA  6, 1         ; SQRT 6,
MULT   6, 6         ;
TAKE   6, 1005      ; +3
DVD    6, 1014      ; E
SUBT   6, 6         ;
TAKE   6, 1008      ; H
ADD    6, 1006      ; I
EXTRA  6, 12        ; INT 6,
PUT    6, *1010(6)  ; G
```

umber of Sheet_____ Name_____

The name need not be written on any sheet after the first

Number Question Y must be n in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

Error detection

The writing of this compiler would have been considerably easier if it had not been necessary to detect and search for any possible error a programmer might make. There are three routines which have been written mainly for the purpose of error detection; BRACKET, EFORMAT and OBRAC. These routines are described separately (pages) but associated with ASSIGN there are at the moment four error messages although it is possible that these may be changed as the program is run and new user errors are detected.

WRONG NUMBER OF BRACKETS

This error message is printed out during the BRACKET routine if:

(i) the brackets do not nest properly.

i.e. A := B+C[D(]

(ii) the open and closed brackets do not match on either side of the 'becomes' sign.

i.e. A[C := D]

(iii) the number of open brackets is not equal to the number of closed brackets.

i.e. A := B(C[D]+3))

or A[B+(C-D] := 3

ERROR AT:

This message is followed by the six characters in the assignment routine which follow the mistake and is

Index Number

umber of Sheet

Name

The name need not be written on any sheet after the first

Number Question Y must be o in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

printed whenever a mistake is made but it is difficult to decide exactly what the user intended.

e.g. A:= B+*C + D[A]*15 results in

ERROR AT: C+D[A] since it is difficult to decide whether the user intended

A:= B+C +....

A:= B*C +....

A:= B+ E*C+ .....

Similarly A:= F ? D-3*C results in

ERROR AT: D-3*C (six characters or upto the 'nudine').

WRONG NUMBER OF ':='

This message is printed out if no 'becomes sign' is found or conversely too many ':=' signs. At the moment this error is found in BRACKET and depends solely upon the number of ':' rather than ':='. Hence

A:-B+C results in ERROR AT: B+C

but A;=B+C results in WRONG NUMBER OF ':='

I am hoping to change this, to make it more logical, if when BRACKET is rewritten to include the commands IF and LOOP.

SPACE OVERFLOW.

This is printed out if the compiled code overruns the protected area. The routine also removes from page 1 any code generated by the current ASSIGN command so that the reentry point of the assembler can be the same.

Number of Sheet _____ Candidate Number _____

Name _____

The name need not be written on any sheet after the first

Number Question Y must be n in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

Examples of some possible errors and the resulting messages

Statement	Routine in which found	Error message.
A:= "&&&&"	RPR +3	ERROR AT: &&&&"
A:=)*(A	BRACKET +33	WRONG NUMBER OF BRACKETS.
SQRT(A):=3	BEGINNING +15	ERROR AT: A):=3
A[B:=C]+E	BRACKET +22	WRONG NUMBER OF BRACKETS
A+B:=C	BEGINNING +15	ERROR AT: B:=C
A:=B:=C	BRACKET +19	WRONG NUMBER OF ':='
A B+C	BRACKET +24	WRONG NUMBER OF ':='
A:=B+*C	EFORMAT +17	ERROR AT: C
A:=B+3C	REDE +11	ERROR AT: .
A:=(B+C)(D+E)	OBRAC +8 ~~EFORMAT +17~~	ERROR AT: D+E)
A[B**C]:=3	EFORMAT +17	ERROR AT: C]:=3
A:=B+()C	EFORMAT +17	ERROR AT: C
A:=B[3](C+D)	OBRAC +10	ERROR AT: C+D)

I have no doubt that users will make other errors as yet undetected but it will be easier to deal with these as they arise.

Number of Sheet_____

Index Number_____

Name_____

The name need not be written on any sheet after the first

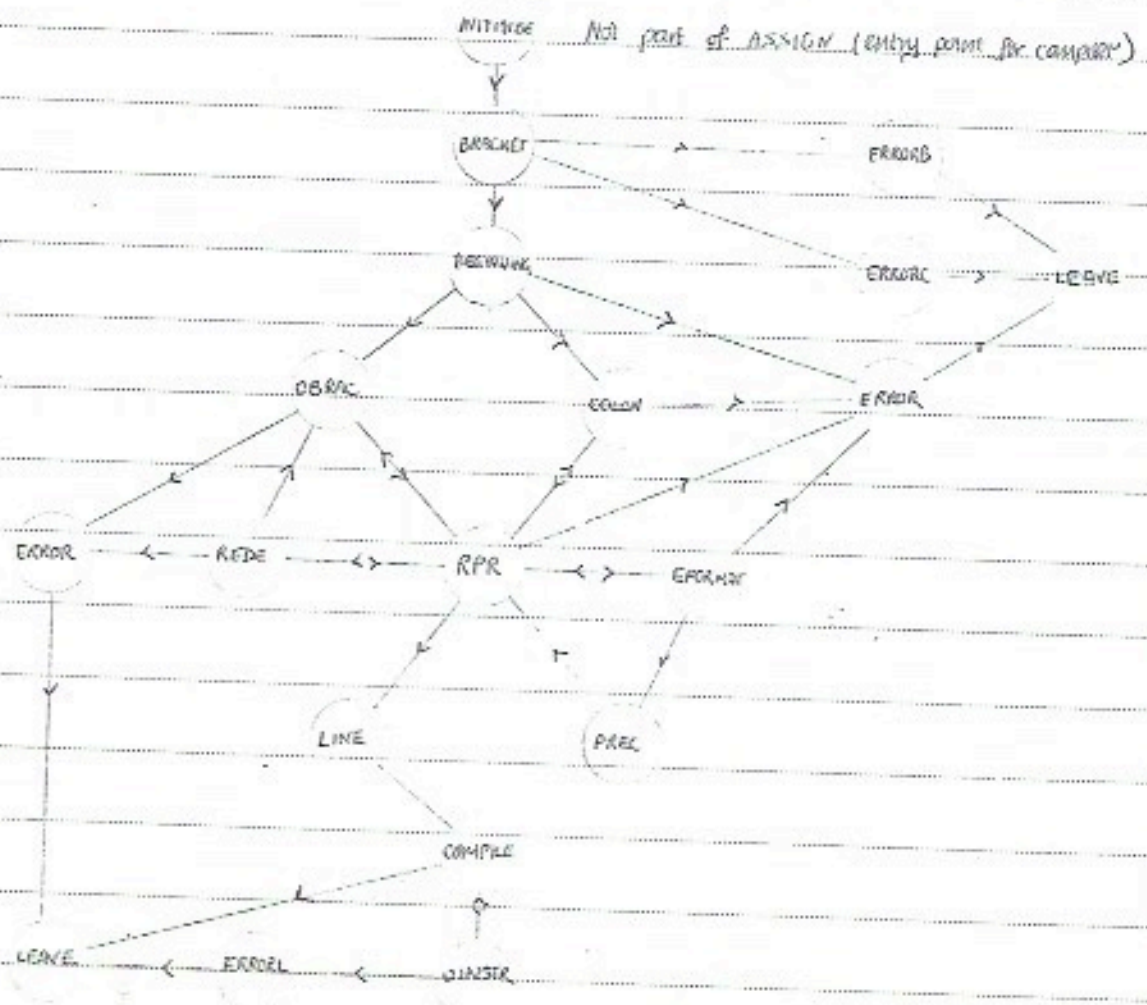
Number Question Y must be in in this in

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

Flowchart and code for 'ASSIGN'

This program is split up into a number of small routines (so that adaption and debugging would be easier) and for this reason the flowchart is split up into a great number of small flowcharts each accompanied by a short description of the purpose of the routine. A diagram showing the connection between these routines is included. I have also included a detailed version of the actual code written in full BBCxas I felt that the code which was input to the machine was extremely difficult to follow particularly since symbolic addressing and literals were not available in the temporary assembler.



Index Number

umber of Sheet

Name

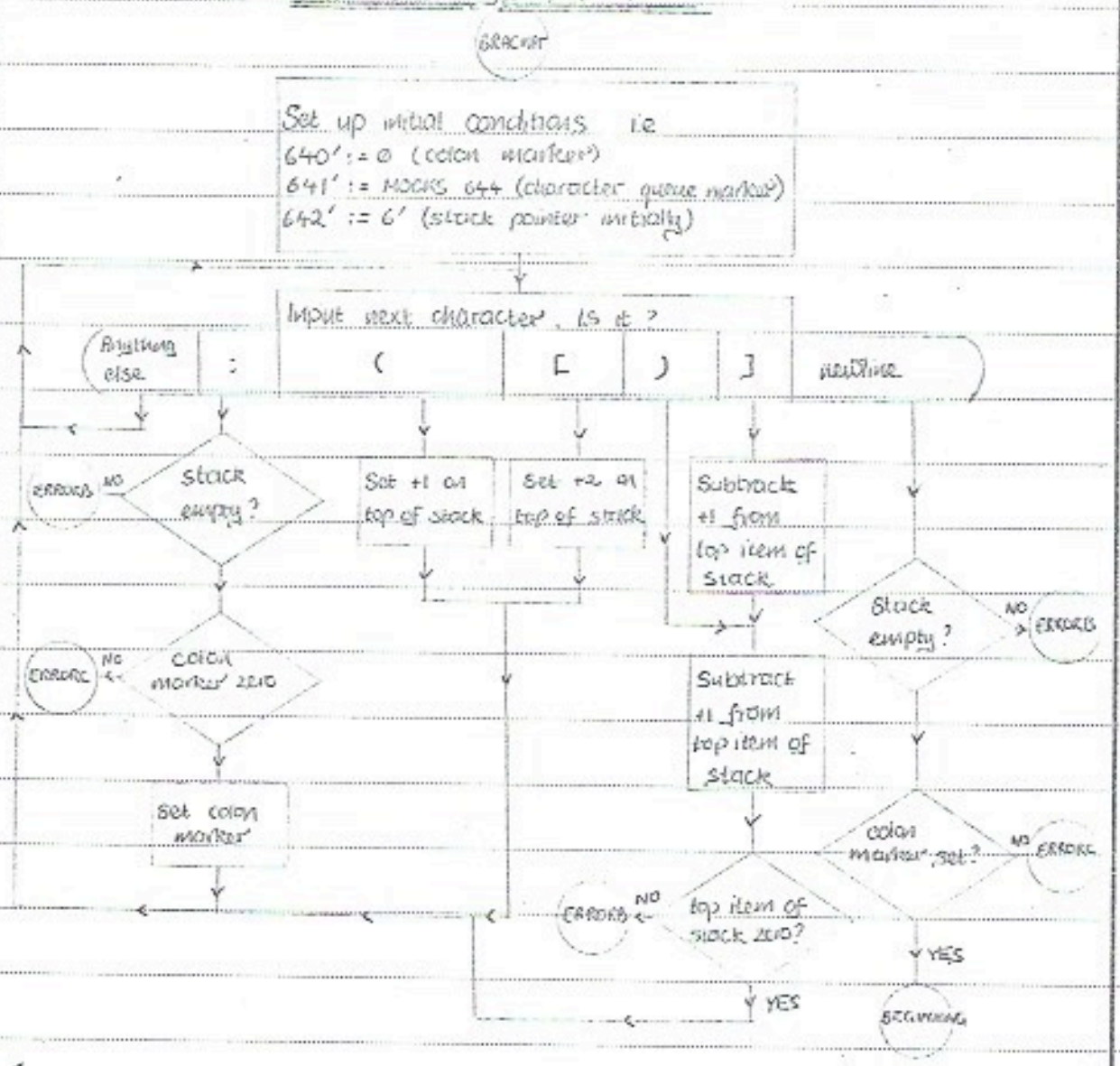
The name need not be written on any sheet after the first

Number Question Y must be n in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

Flow chart for ASSIGN



This is the flowchart for the routine 'BRACKET' and in an initial scan checks that the parentheses are of the correct number and are correctly nested. It also checks that there is only one colon. This routine makes great use of the hardware stack.

e.g. D:=A[(B]+C) results in an error.

D := (A+B)) " " " "

D[:= C] " " " "

D := A : B " " " "

Index Number_____

Number of Sheet_____

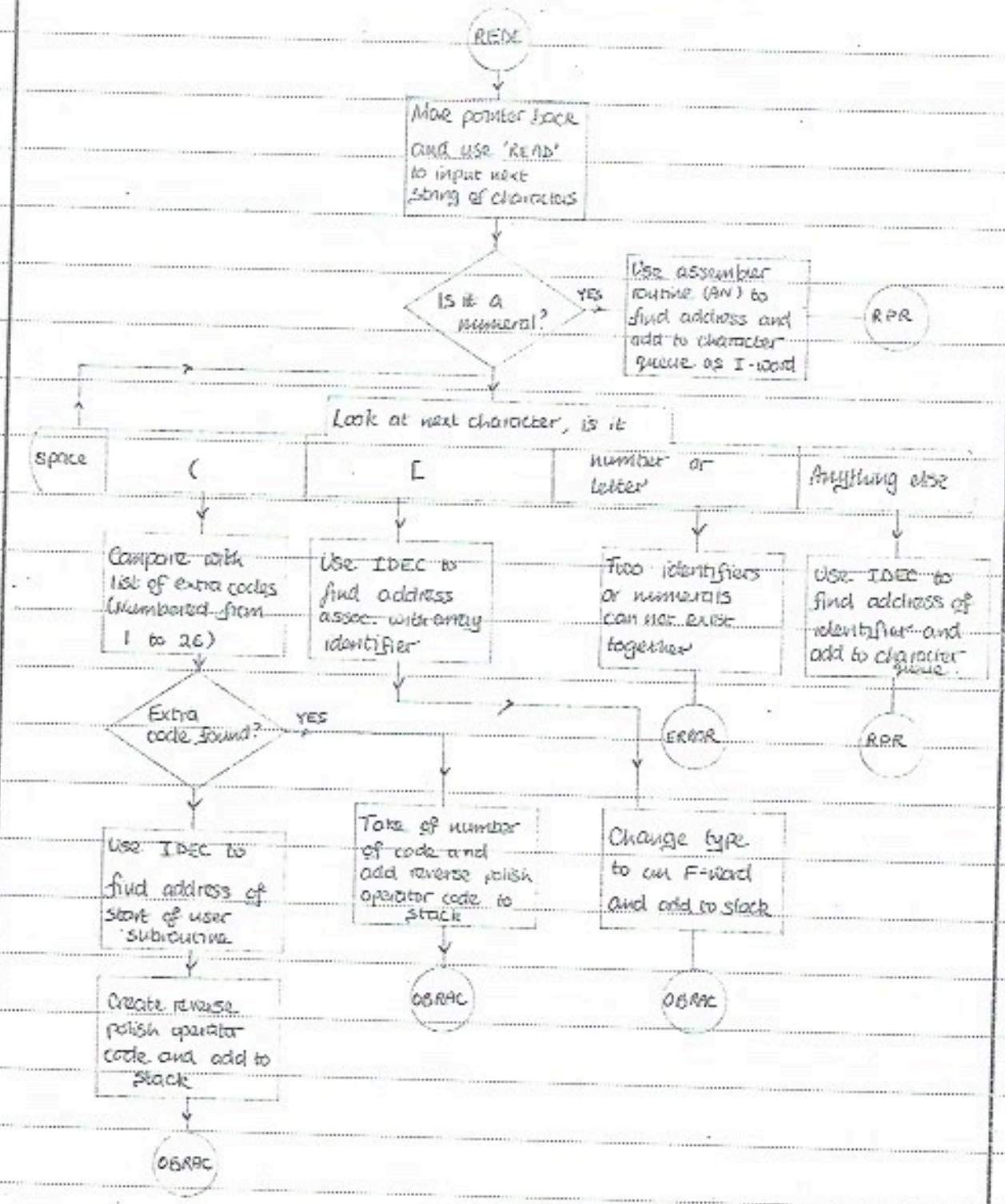
Name_____

The name need not be written on any sheet after the first

Number Question Y must be in in this

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank



This routine deals with all identifiers. The BECX extra codes 'READ' is capable of inputting both constants and identifiers. Identifiers may be any number of characters long, but must begin with a letter. 'READ' follows the usual 'ALGOL' convention and thus the result can directly be used as a parameter for the assembler routines IDEC and AN. IDEC adds a new identifier into the protected area

Index Number _____

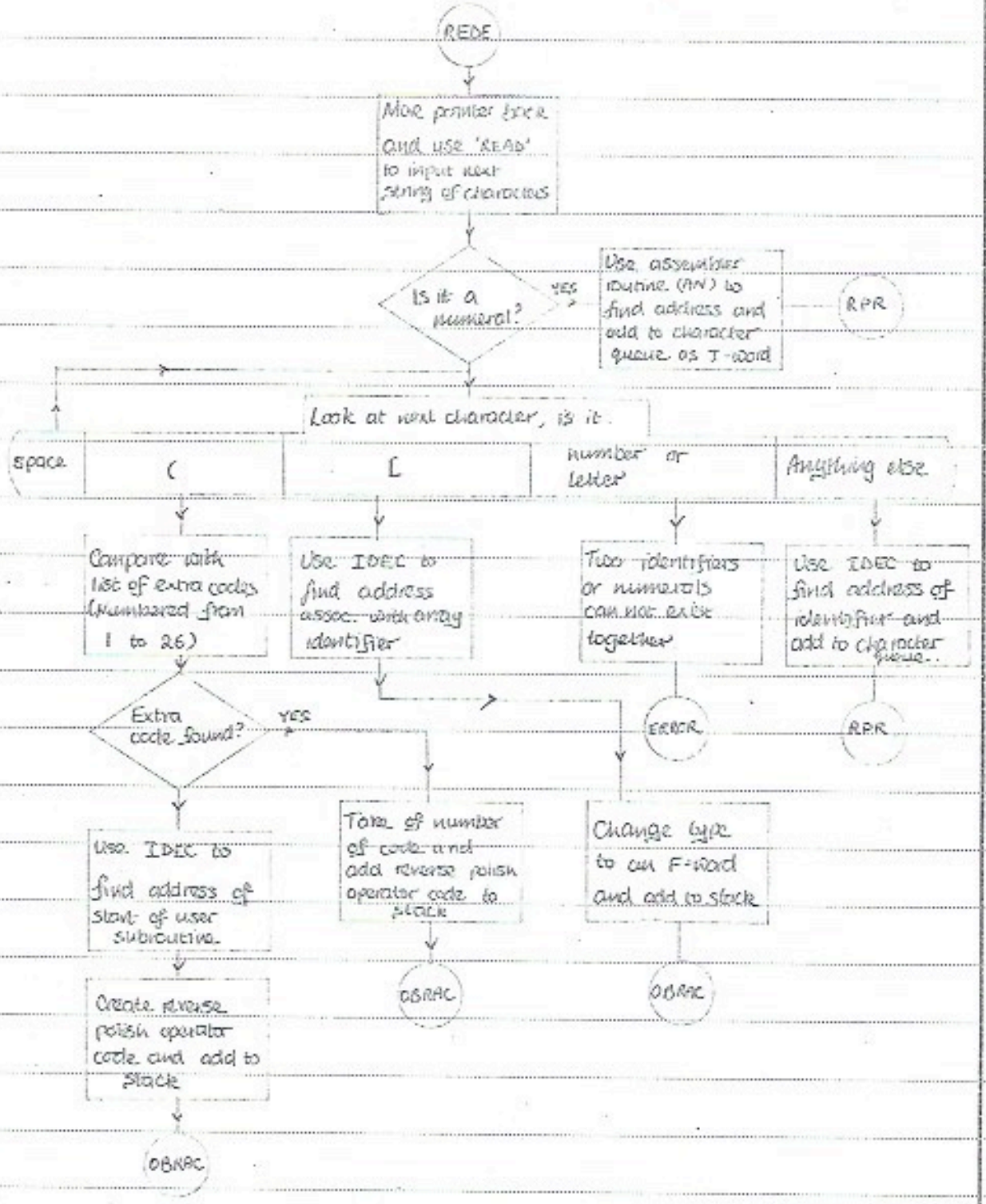
umber of Sheet _____ Name _____

The name need not be written on any sheet after the first

Number Question Y must be n in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank



This routine deals with all identifiers. The BECX extra codes 'READ' is capable of inputing both constants and identifiers. Identifiers may be any number of characters long, but must begin with a letter. 'READ' follows the usual 'ALGOL' convention and thus the result can directly be used as a parameter for the assembler routines IDEC and AN. IDEC adds a new identifier into the protected area

umber of Sheet _____ Name _____

The name need not be written on any sheet after the first

Number Question f must be n in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

(and outputs a warning message) if nessessary. The actual name of the identifier is stored in an odd cell and reserves associated workspace in the even cell preceeding it in the store. eg 1019 contains "AABbbb"

1018 contains associated workspace.

It is this even address that is returned by the routine IDEC.

AN stores the constant in a cell (if it has not been used before) and returns the address of this cell.

N.B. the function of the assembler with regard to identifiers and constants is explained in more detail in the specification in the section called 'The Assembler'

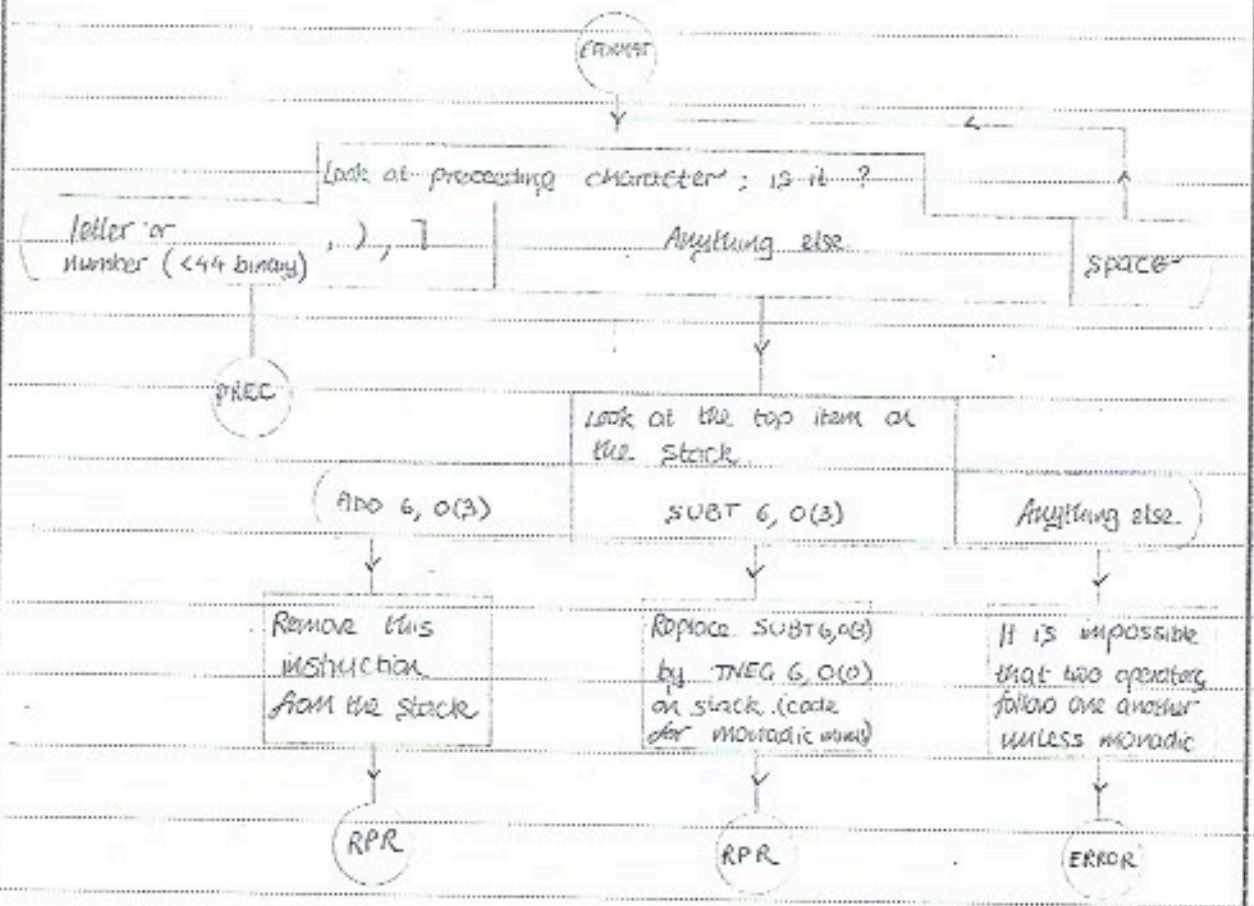
Number of Sheet _____ Name _____

The name need not be written on any sheet after the first

Number Question Y must be n in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank



This routine is used when an operator (in the P-word group (+, −, /, *, ↑) or a closed parentheses (),]) is input. It checks to ensure that none of these characters follow +, −, /, *, ↑, (, [, = since this would result in an error. There is however an exception to this, when + or − refer to ⊕ or ⊖ (i.e. monadic operators). In this case the routine ignores a monadic + and compiles code for a monadic −. Thus this routine differentiates between ⊖ and − without the use of brackets.

e.g. A+*B results in an error

(A+B)(B+C) results in an error.

but B++A or B*−A do not result in an error.

but B−*A ~~d~~ results in an error

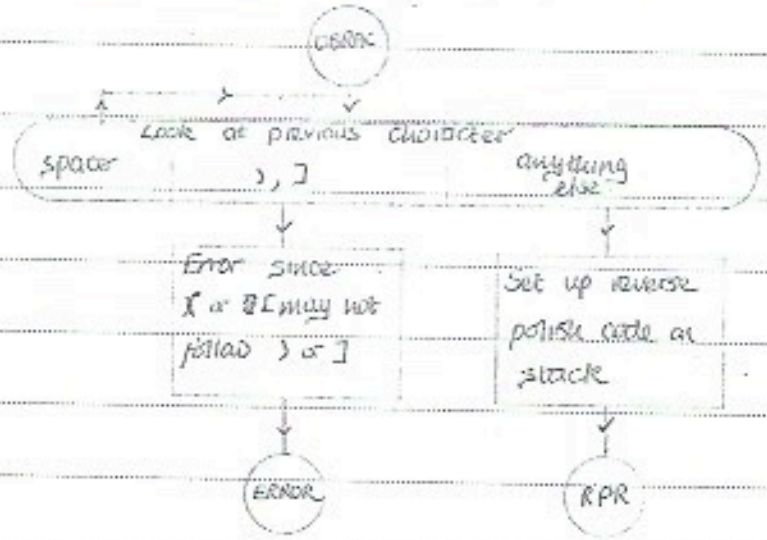
Number of Sheet _____ Name _____

The name need not be written on any sheet after the first

Number Question Y must be n in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank



This routine is entered whenever '(' or '[' is input. Initially it tests to ensure that '(' or '[' has not followed ')' or ']' Thus (A+B)(C+D) results in an error but not (A+B)*(C+D). The routine also sets up the reverse polish operator code on the stack. There is no need to enter PREC (the routine for checking precedence codes) since ~~for~~ open brackets act as a stopper on the stack and characters cannot be unloaded beneath them until a closed bracket is encountered.

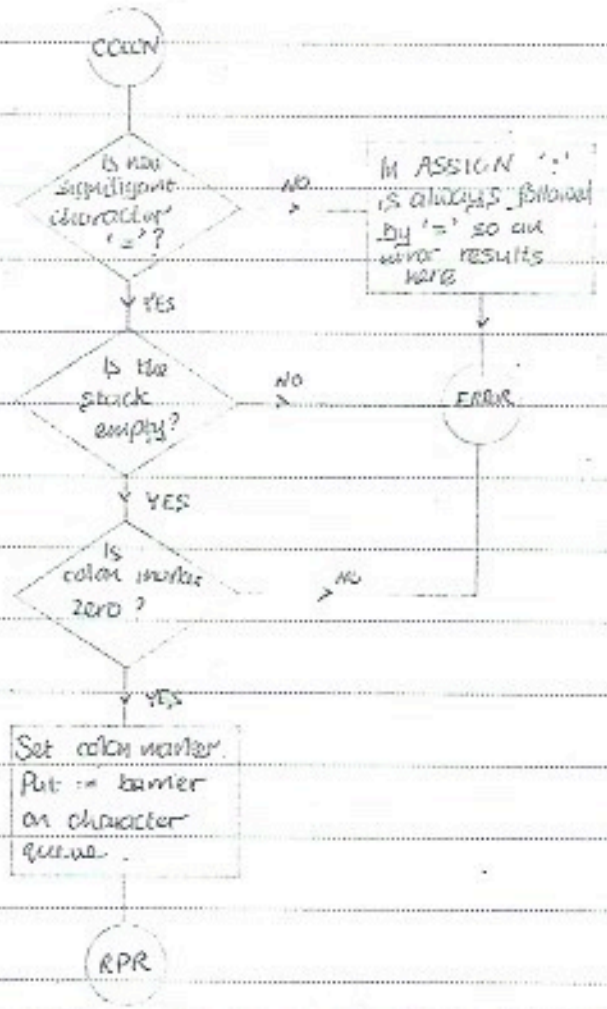
Number of Sheet _____ Name _____

The name need not be written on any sheet after the first

Number Question Y must be n in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank



A great part of this routine is superfluous for ASSIGN but when IF and LOOP are implemented it will become more important. It is unnecessary to test if the stack is empty since BRACKET ensures that brackets are correctly nested on either side of ':=' and BEGINNING tests to ensure that first ~~the~~ character after destination identifier is ':' or '['. Similarly the colon marker will be zero since BRACKETS checks colons. However it is thought that ~~both~~ BRACKETS will have to be considerably altered for IF and LOOP and neither will use BEGINNING of course.

However the routine does check that the colon is followed by 'equals' (they may be separated by spaces)) and puts a colon marker on the reverse polish character queue.

Index Number_____

umber of Sheet_____

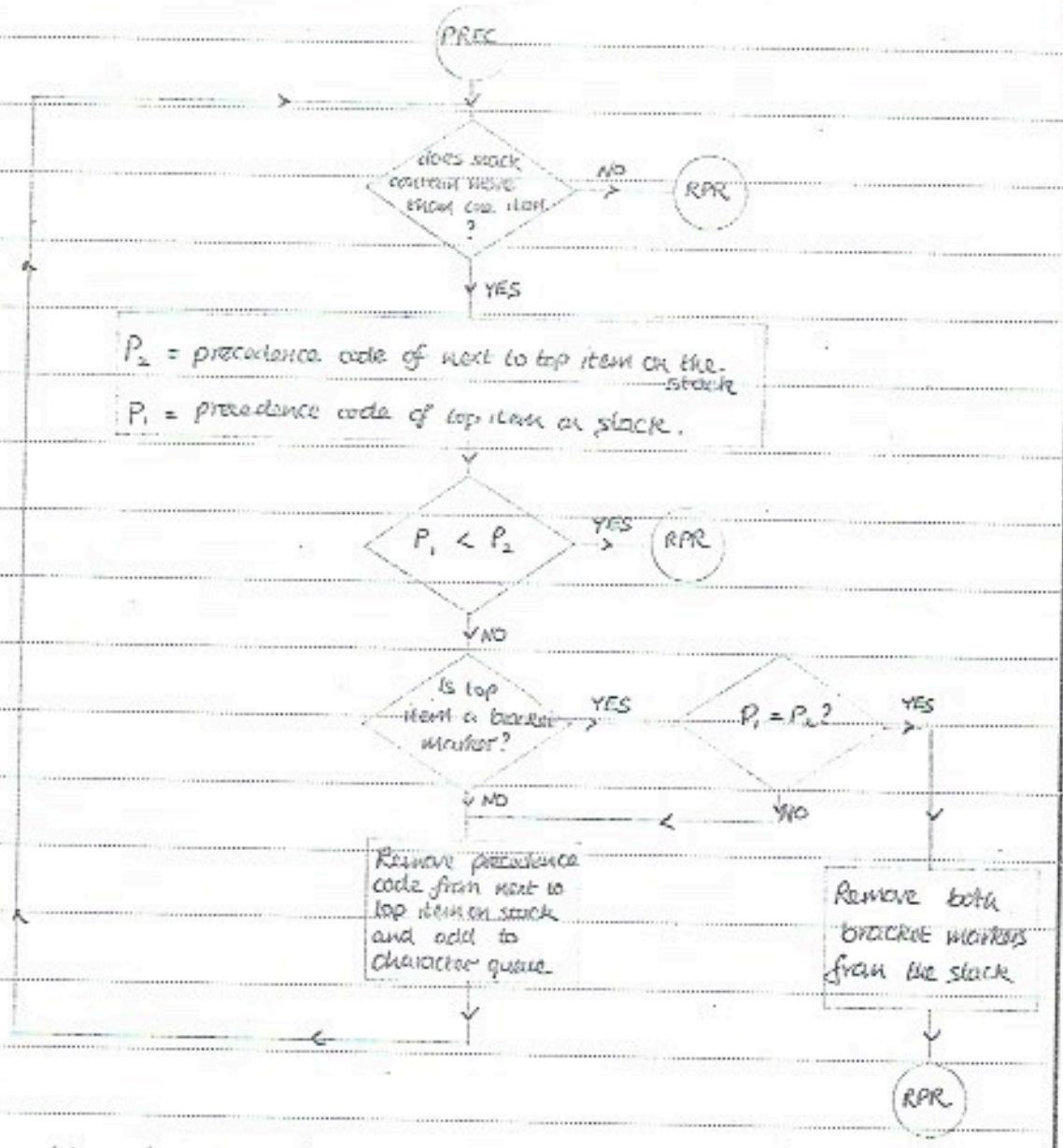
Name_____

The name need not be written on any sheet after the first

Number Question y must be n in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank



This routine is entered whenever +, −, ↑, *, /,),] are added to the stack and using the precedence codes unloads operators of lower or equal precedence. If) or] are added everything will be removed down to (or [and both markers are then discarded.

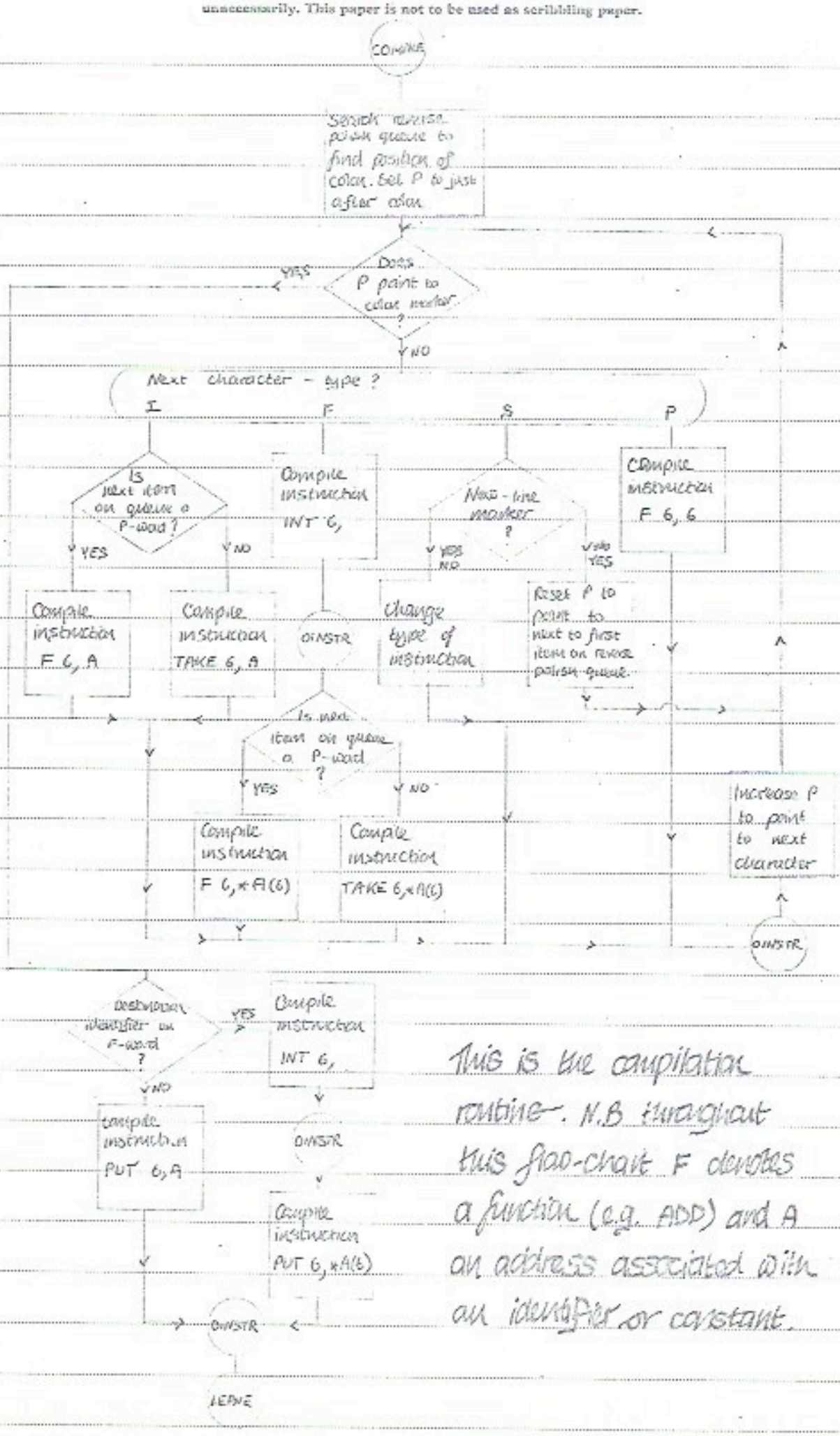
umber of Sheet_____ Name_____

The name need not be written on any sheet after the first

Number Question Y must be n in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank



This is the compilation routine. N.B throughout this flow-chart F denotes a function (e.g. ADD) and A an address associated with an identifier or constant.

umber of Sheet_____ Name_____

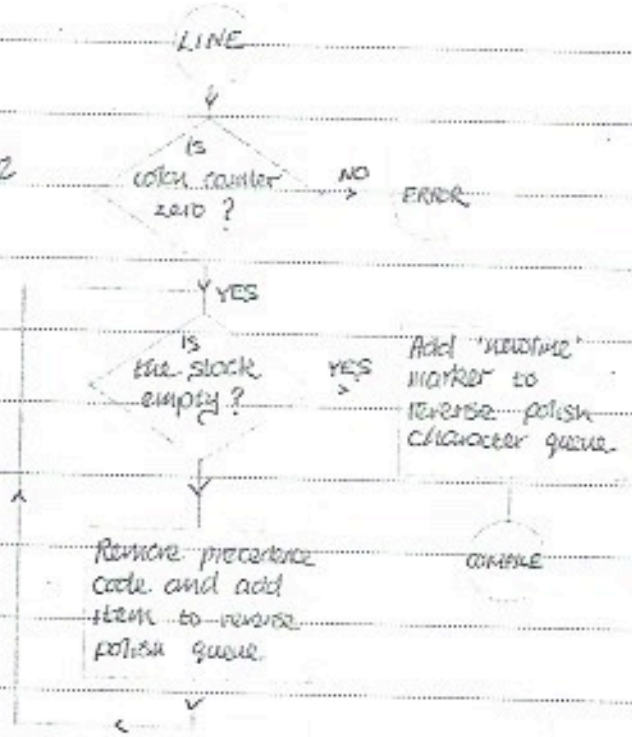
The name need not be written on any sheet after the first

Number Question Y must be n in this n

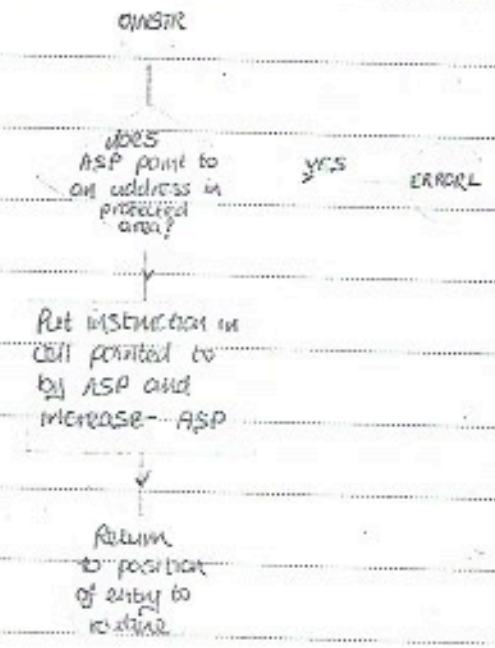
Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

This should not be needed in ASSIGN.



This routine unloads the stack when a newline is input denoting the end of the arithmetic expression.



ASP - Assembly storage pointer.

This routine puts an instruction into the end of the user's program first checking that this will not run into the protected area (ie. the storage of identifiers with their workspace and constants)

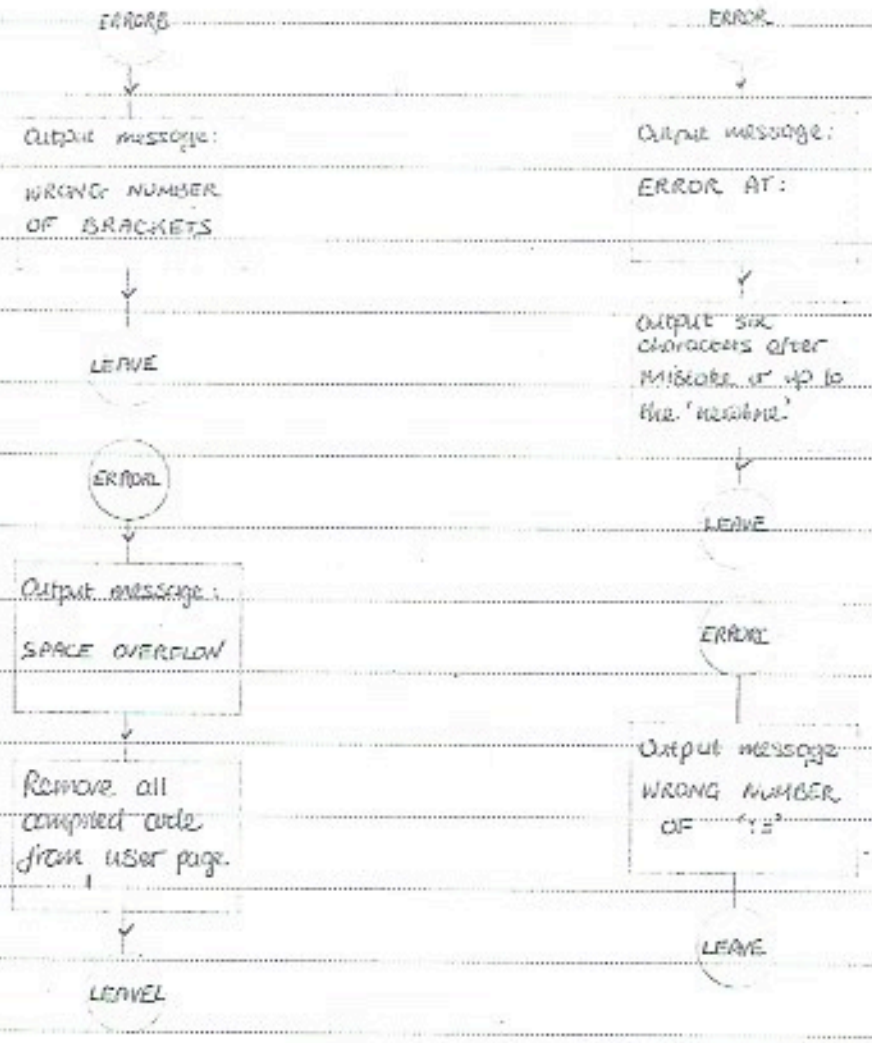
umber of Sheet_____ Name_____

The name need not be written on any sheet after the first

Number Question must be n in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank



ERRORB, ERROR, ERRORL and ERRORC are routines for outputting error messages.

LEAVE is a routine for departing from ASSIGN, it leaves the stack as it was on entry and updates ASP.

The name need not be written on any sheet after the first

Number Question must be in this

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

```
INITIALISE : PUT Ø, 8          ; set page bit.
             1/ PUT 6, 895'    ; copy of return address in 895'
             2/ PUT 6, 894'    ; copy of command code in 894'
             3/ TAKE 1, 894'
                EXEC  LOOK(1)
             4/ JUMP * LOOK(1) ; jumps to a different part of
                               the program depending upon
                               the command input.

[LOOK : 2/ JUMP * BRACKET      ; the code for ASSIGN is 2.]
Code for ASSIGN
39 BRACKET : PUT Ø, 8
             1/ TAKE 3, 6'     ; Copy of stack pointer in 642' so
             2/ PUT 3, 642'    ; that stack can be 'emptied'
             3/ TAKE 6, [MOCKS 644] ; 641' contains a pointer for the
             4/ PUT 6, 641'    ; reverse polish character queue.
             5/ PUT Ø, 64Ø'    ; colon marker
             6/ PUT Ø, 4       ; copy of colon marker for BRACKET.
             7/ TAKE 6, 1'     ; copy of input pointer in 643'
             8/ PUT 6, 643'
             9/ TAKE 6, Ø      ; base for bracket markers
             1Ø/ TAKE 3, * 643' ; look at next character.
             11/ SUBT 3, +46   ; acc 3 = Ø if character was '('
             12/ JLZ 3, 19/    ; number or letter input.
             13/ SKAB 3, +3    ; skip if a bracket was not input
             14/ JUMP 1, 33/   ; bracket input.
             15/ SUBT 3, +16   ; newline ?
             16/ JEZ 3, 24/
             17/ ADD 3, +4
```

Number Question Y must be n in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

```
18/ JNZ  3, 10/          ; acc 3 = Ø if acdon has been
                         ; input
19/ JNZ  4, * ERRORC     ; too many colons in assignment
                         ; statement.
20/ INCR  Ø, 4           ; set colon marker.
21/ SKAE  6, Ø           ; is stack empty?
22/ JUMP  * ERRORB       ; brackets do not nest on each
                         ; side of 'becomes' sign.
23/ JUMP  10/            ; repeat bracket testing process.
24/ JEZ  4, * ERRORC     ; no colon in assignment statement
25/ SKAE  Ø, 6           ; is top stack empty? (pops).
26/ JUMP  * ERRORB       ; not enough closing brackets.
27/ JUMP  * BEGINNING    ; leave BRACKET routine.
28/ SUBT  6, +1          ; these remove bracket markers.
29/ SUBT  6, +1          ;
30/ JNZ  6, * ERRORB     ; brackets incorrectly nested.
31/ PUT  6, Ø            ; remove marker of this bracket
                         ; loop.
32/ JUMP  10/            ; reenter BRACKET routine.
33/ EXEC  BRAC(3)        ; the elements of BRAC obey
34/ JUMP  10/            ; different instructions depending
                         ; upon the type of parenthesis
                         ; input.
BRAC  : TAKE  6, +1      ; marker for '('
  1/ JUMP  257/          ; jump to routine for ';'
  2/ TAKE  6, +2         ; marker for '['
  3/ JUMP  28/           ; jump to routine for ']'
```

Number Question must be in this

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

```
BEGINNING : READ   4,4          ;input destination identifier.
            1/ TAKE  2,*1'      ;look at next character to see
                                ;if destination is an array
                                ;element.
            2/ SUBT  2,+61      ;to test for a space.
            3/ JEZ   2, 1/      ;spaces are ignored
            4/ ADD   2, +13     ;acc 2 = Ø => 'L' input.
            5/ JNZ   2, 14/     ;simple identifier as destination
            ~~6/ TAKE  6, ASS~~   ;} this is now pointing to the   To enter
               JUMP    7, * ENTER
            ~~7/ PUT   6, 8~~     ;} assembler page (stored in ASS)  IDEC.
            ~~8/ JUMP  7, *IDEC~~ ;assembler routine which has
                                ;as input an S-word in acc.3
                                ;and as output an I-word
                                ;address in acc 3.
            9/ TAKE   6,+1      ;} change type of address to
            10/ PTYP  6, 4      ;} an F-word.
            ~~11/ PUT  Ø, 8~~     ;~~change page back to user Ø.~~
            12/ PUT   4,*641'   ;store array identifier as first
                                ;item on character queue.
            13/ JUMP  * OBRAC   ;enter reverse polish routine.
            14/ SUBT  2, +10    ;is character ':' ?
            15/ JNZ   2, * ERROR ;':' or 'L' must follow first
                                ;identifier.
            ~~16/ TAKE  6, ASS~~  ;} changes page bit to   To enter
               JUMP    7, * ENTER
            ~~17/ PUT  6, 8~~     ;} point to assembler.   IDEC
            ~~18/ JUMP  7, * IDEC~~ ;enter assembler routine.
            ~~19/ PUT  Ø, 8~~     ;change page back to user Ø
            20/ PUT   4,*641'   ;store identifier.
```

Number Question must be in this margin

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

```
21/ JUMP  * COLON          ; enter reverse polish routine.

RPR  : TAKE 2, *1'         ; inputs as an I-word the
                           ; next character from the buffer
                           ; into L.S.E of acc. 2.
     1/ SUBT  2, +44       ; +44 is decimal value of ':'
                           ; (the first non-readable character)
     2/ JLZ   2, * REDE    ; REDE is a routine to cope
                           ; with identifiers, extra codes
                           ; user labels and numbers.
     3/ EXEC  I (2)        ; this treats all other
                           ; characters individually
     4/ JUMP  * EFORMAT    ; this jump is used by seven
                           ; characters viz: +, -, /, *, ↑, ), ]

I: TAKE  6, [ADD 6, Ø(3)]; 3 is precedence code for '+'
1/ TAKE  6, [SUBT 6, Ø(3)]; 3 " " " " '-'
2/ JUMP  * OBRAC           ; '(' has been input
3/ TAKE  6, (IØØØ4ØØØØ); 4 is precedence code for ')'
4/ JUMP  * OBRAC           ; '[' has been input
5/ TAKE  6, (IØØØ4ØØØØ); 4 is precedence code for ']'
6/ TAKE  6, [MULT 6, Ø(2)]; 2 " " " " '*'
7/ TAKE  6, [DIV 6, Ø(2)]; 2 " " " " '/'
8/ JUMP  * ERROR           ; '=' should never occur since
                           ; if ':' is input '=' is detected
                           ; and ignored in routine colon.
9/ JUMP  * ERROR           ; '≠' should not occur in ASSIGN
10/ JUMP * ERROR           ; '←' " " " " "
```

Number Question must be in this

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

```
11/ TAKE 6, [.POWR 6, Ø(1)  ; ↑ is precedence code for '↑'
12/ JUMP *ERROR             ; '?' should not occur in ASSIGN
13/ JUMP *ERROR             ; (") " " " " "
14/ JUMP *COLON             ; subroutine to process ':'
15/ JUMP *ERROR             ; ';' should not occur in ASSIGN
16/ JUMP *ERROR             ; ',' " " " " "
17/ JUMP *RPR               ; spaces are ignored.
18/ JUMP *LINE              ; newline is the terminating
                            ; character for ASSIGN.

REDE : DECR Ø, 1'           ; move pointer to beginning of word
1/ READ 4/3,                ; S-word or numeral in acc. 3
                            N.B. numerals are now detected by IDEC
~~2/ JAP 3, *NO~~           ; ~~numeral has been read.~~
3/ TAKE 2, *1'              ; look at next character.
4/ SUBT 2, +61              ; } spaces are ignored.
5/ JEZ 2, 3/                ;
6/ ADD 2, +15               ; '(' ?
7/ JEZ 2, *LOE              ; label or extra code.
8/ SUBT 2, +2               ; '[' ?
9/ JEZ 2, *ARRAY            ; array identifier.
10/ ADD 2, +4               ; } two numbers or identifiers
11/ JLZ 2, *ERROR           ; } may not occur together.
12/ DECR Ø, 1'              ; move pointer back.
~~13/ TAKE 6, ASS~~         ; } change page to point to
JUMP 7, *ENTER              ;
~~14/ PUT 6, 8~~            ; } assembler.
~~15/ JUMP 7, *IDEC~~       ; assembler routine to find address
~~16/ PUT Ø, 8~~            ; change back to user page Ø
17/ PUT 4/3, *641'          ; store identifier
```

To enter IDEC.

umber of Sheet 38 Name

The name need not be written on any sheet after the first

Number Question must be in this margin

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

```
      18/ JUMP * RPR          ; enter reverse polish routine.
LOE:  19/ TAKE 3, +1          ; counter and code
      20/ TAKE 2, +27         ; there are 26 extra codes.
      21/ SKAN 4, E(3)        ; skip if extra code not located.
      22/ JUMP * EXTRA        ; extra code located.
      23/ INCR Ø, 3           ; increase counter
      24/ SKAE 2, 3           ; skip if end of extra code search
      25/ JUMP 21/            ; repeat search for extra code.
      26/ TAKE 6, ASS         ; change page
          JUMP 7, * ENTER
      27/ PUT 6, 8            ;                                  to enter
      28/ JUMP 7, * IDEC      ; this is a user subroutine to     IDEC.
                              ; find address of start.
      29/ TAKE 6, (56Ø7Ø4ØØØ); JUMP 7, * Ø(Ø) as an S-word.
      30/ OR 6, 4             ; JUMP 7, * A(Ø) as an S-word.
      31/ PUT Ø, 8            ; change page back.
      32/ JUMP * OBRAC        ; enter reverse polish routine.
EXTRA: 33/ TAKE 6, (577 6ØØØØØ); EXTRA 6, Ø as an S-word.
      34/ OR 6, 3             ; acc. 3 contains extra code
                              ; mock address.
      35/ JUMP * OBRAC        ; enter reverse polish routine.
NO:   36/ TAKE 2, * 1         ; input next character.
      37/ SUBT 2, +61         ;
      38/ JEZ 2, * NO         ; spaces are ignored.
      39/ ADD 2, +17          ;
      40/ JLZ 2, * ERROR      ; A number may not be followed
                              ; by another number or set of
                              ; letters.
      41/ TAKE 6, ASS         ; change page.
      42/ PUT 6, 8            ;
```

Number of Sheet 39

Index Number

Name

The name need not be written on any sheet after the first

Number Questions Y must be n in this in

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

```
43/ JUMP 7, * AN          ; assembler routine to find an
                          ; address for a constant. Number
                          ; is input to routine in acc. 3 &
                          ; address is returned to 3 as an
                          ; F-acc.
44/ PUT 0, 8              ; zero page ~~int~~ cell.
45/ PUT 3, * 641'         ; store address of constant.
46/ JUMP * RPR            ; enter reverse polish routine.
ARRAY: ~~47/ TAKE 6, ASS~~  ;} change to assembler page.
~~48/ PUT 6, 8~~            ;}
       JUMP 7, * ENTER
~~49/ JUMP 7, * IDEC~~      ; assembler routine to find address
~~50/ PUT 0, 8~~            ; zero page cell.
51/ TAKE 6, +1            ;} change type of address to an
52/ PTYP 6, 3 4           ;} F-acc.
53/ PUT 4 3, 6            ; put on reverse polish stack.
54/ JUMP * OBRAC          ; enter reverse polish routine.

E: 1/ "SQRT"              12/ "INT&"
   2/ "LN&&"              13/ "FRAC"
   3/ "EXP&"              14/ "FLOT"
   4/ "READ"              15/ "CAPN"
   5/ "PRIT"              16/ "PAGE"
   6/ "SIN&"              17/ "RND&"
   7/ "COS&"              18/ "ABS&"
   8/ "TAN&"              19/} "&&&&" ; these are in case
   9/ "ATN&"              20/}        ; new extra codes
   10/ "STOP"                         ; are introduced.
   11/ "LINE"
```

To enter IDEC

180

```
ENTER : TAKE 6, ASS        ; } Change to assembler page.
        1/ PUT 6, 8        ;
        2/ TAKE 6, 2       ; stack accumulator 2 to ensure content is not lost.
        3/ JUMP 7, *IDEC   ; enter assembler routine to find address of constant.
        4/ PUT Ø, 8        ; clear page bit.
        5/ PUT 6, 2        ; unstack accumulator 2.
        6/ JUMP *6         ; return.
```

N.B. Transfer character i.e. parameter of IDEC enters in acc. 4.

Number of Sheet 40

Index Number

Name

The name need not be written on any sheet after the first

Number Question Y must be in this m

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

```
EFORMAT : TAKE  6, 1'          ; Puts a copy of the input
          1/ PUT   6, 643'     ; buffer pointer in a cell in page Ø
          2/ DECR  Ø, 643'     ; pointer now indicates the
          3/ DECR  Ø, 643'     ; character before the one just
                               ; input.
          4/ TAKE  2, *643'    ; take this character to acc 2.
          5/ SUBT  2, +44      ; tests for a preceeding letter
          6/ JEZ   2, *PREC.   ; or number.
          7/ SUBT  2, +3       ; tests for a preceeding ')'
          8/ JEZ   2, *PREC    ;
          9/ SUBT  2, +2       ; tests for a preceeding ']'
          10/ JEZ  2, *PREC    ;
          11/ SUBT 2, +12      ; spaces are ignored.
          12/ JEZ  2, 2/       ;
          13/ TAKE 2, 6        ; unload top item of reverse
                               ; polish stack into acc 2.
          14/ AND  2, (P73147777); this is NOT of code stored for
                               ; '+' and therefore 2 will only
                               ; be zero if instr. was ADD 6,Ø(3)
                               ; i.e. (P0463ØØØØ)
          15/ JEZ  2, *RPR     ; i.e '+' was monadic in this case
          16/ AND  2, (P76777777); acc 2 will be zero if and
                               ; only if '-' was preceeding character
          17/ JNZ  2, *ERROR   ; '+' and '-' are the only two
                               ; monadic functions in the
                               ; group '*', /, +, -, ↑'
          18/ TAKE 6, (P12CØØØØØ) ; TNEG 6,Ø added to reverse
                               ; polish stack.
```

Number Question must be in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

```
19/ JUMP * RPR          ; enter reverse polish routine.

OBRAC : TAKE 6, 1'      ;
        1/ PUT 6, 643'  ; Copies input buffer pointer
                        ; to 643 in page ø.
        2/ DECR ø, 643' ;
        3/ DECR ø, 643' ; to look at previous character
        4/ TAKE 2, *643' ;
        5/ SUBT 2, +61  ; spaces are ignored.
        6/ JEZ 2, 2/    ;
        7/ ADD 2, +18   ; an open bracket may not
        8/ JEZ 2, *ERROR ; follow ')'
        9/ SUBT 2, +2   ; an open bracket may not
        10/ JEZ 2, *ERROR ; follow ']'
        11/ TAKE 6, (IØØØ4ØØØØ); precedence code for '(' or '[' ;
                        ; into reverse polish stack.
        12/ JUMP * RPR  ; return to reverse polish routine.

COLON : TAKE 2, *1'     ; look at next character to find '='
        1/ SUBT 2, +61  ; spaces are ignored.
        2/ JEZ 2, *COLON ;
        3/ ADD 2, +9    ; only a space or '=' can
        4/ JNZ 2, * ERROR ; follow ':' in ASSIGN.
        5/ TAKE 2, 642' ; copy of stack pointer initially.
        6/ SKAE 2, 6'   ; the stack should be empty
        7/ JUMP * ERROR ; as arithmetic expressions on
                        ; each side of ':=' are distinct.
        8/ TAKE 2, 640' ; two ':=' in one statement.
        9/ JNZ 2, *ERRORC ;
```

Name

The name need not be written on any sheet after the first

Number Question must be in this margin

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

```
10/ INCR  0, 640'          ; set ':=' marker
11/ TAKE  6, (500000000)   ; set ':=' barrier on reverse polish
12/ PUT   6, *641'         ; stack.
13/ JUMP  *RPR             ; repeat reverse polish routine.

PREC. :  TAKE 3, 6         ; unload top item into 3.
1/ TAKE  2, 6'             ; did stack only have one item
2/ SKAN  2, 642'           ; on it?
3/ JUMP  14/               ; replace top item and return.
4/ PUT 6, 2.               ; unload preceeding item without
                           ; 'popping'.
5/ TNEG  4, *[MOCKP 5,*2(3)]; take as an I-word the
                           ; index register field of acc. 2,
                           ; and negate.
6/ ADD   4, *[MOCKP 5,*3(3)]; add the index register field of
                           ; acc. 3 leaving 3 unchanged.
7/ JLZ   4, 12/            ; top item is of lower precedence.
8/ JAT   3, 16/            ; bracket marker as top item.
9/ AND   2, (177707777)    ; masks out precedence code.
10/ PUT  2, *641'          ; unload stack into character
                           ; queue.
11/ JUMP 1, 1/             ; repeat precedence testing.
12/ TAKE 6, 2              ; replace top item on stack
~~13/ JUMP *RPR~~          ; return to reverse polish routine.
14/ TAKE 6, 3
~~14/ SKET 3, +2~~         ; is does 3 contain an I-word.
15/ JUMP *RPR.
~~15/ JUMP 9/~~            ; F-word in 3 (array identifier).
16/ SKET 3, +2
~~16/ JNZ 2, 9/~~          ; open bracket marker not yet
                           ; encountered.
```

```
PREC : TAKE  3, 6                  ; unload top item into 3.
       1/ TAKE  2, 6'              ; } did stack only have one item on it ?
       2/ SKAN  2, 642'            ;
       3/ JUMP  14/                ; to replace top item & return.
       4/ PUT   6, 2               ; unload preceeding item on stack.
       5/ TNEG  4, * [MOCKP5,*2(3)]; take as an I-word the index register
                                   ; field of acc. 2 & negate.
       6/ ADD   4, * [MOCKP5,*3(3)]; add the index field of acc. 3 to
       7/ JLZ   4, 12/             ; top item is of lower precedence.
       8/ JAT   3, 15/             ; bracket marker ~~at st~~ as top item or F-word. possibly
       9/ AND   2, (I77707777)     ; mask out precedence code.
       10/ PUT  2, * 641'          ; unload stack into character queue.
       11/ JUMP 1, 1/              ; repeat precedence testing.
       12/ TAKE 6, 2               ; replace lower item on stack.
       13/ TAKE 6, 3               ; replace upper item on stack.
       14/ JUMP * RPR              ; reenter reverse polish routine
       15/ SKET 3, +2              ; open bracket marker not yet encountered.
       16/ JUMP 1, 9/              ; unload item
       17/ JNZ  4, 9/              ; F-w
       15/ SKET 3, +2              ; F-word or I-word in acc. 3 ?
       16/ JUMP 1, 9/              ; F-word (i.e. array identifier)
       17/ JNZ  4, 9/              ; open bracket marker not yet encountered.
       18/ JUMP 1, * RPR           ; re-enter routine (both bracket markers
                                   ; are automatically lost)
```

Number Question Y must be in in this in

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

```
                JUMP 1, 9/
17/ ~~DEER 0,6'~~          ; this in effect eliminates
                           ; open bracket marker.
                JNZ 4, 9/
18/ ~~JUMP *RPR~~          ; return to reverse polish routine.
19/ JUMP 1, *RPR

LINE  ; TAKE 2, 640'       ;} no ':=' found in ASSIGN.
      1/ JEZ 2, *ERRORC    ;}
      2/ TAKE 2, 642'      ;} 642' is initial position of stack
      3/ SKAN 2, 6'        ;} pointer ⇒ skip if stack is not
                           ; empty.
      4/ JUMP 8/           ; stack is empty.
      5/ AND 6, (I7770777) ; mask out precedence code of top
                           ; item.
      6/ PUT 6, *641'      ; unload into character queue.
      7/ JUMP 3/           ; repeat unloading process.
      8/ TAKE 6, (S00000001);} put terminating character on
      9/ PUT 6, *641'      ;} reverse polish queue.
      10/ JUMP *COMPILE    ; jump to routine for code
                           ; formation.

COMPILE ; TAKE 5, +2       ; to change type to a P-word.
        1/ TAKE 4, 11'     ; copy of assembly stack pointer (MOCKP word).
→          OR  4, [NIL 0,0'] ; to set page bit on A.S.P.
        2/ TAKE 6, [MOCKS 645] ;} character ~~stack~~ queue pointer
        3/ PUT 6, 641'     ;} initially set to next to first
                           ; character.
        4/ TAKE 2, *641'   ; look at next character.
        5/ JAT 2, 4/       ;} Searching for (S00000000) (:= marker)
        6/ JNZ 2, 4/       ;} care must be taken not to find (I00000000).
```

Number Question You must be in in this margin

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

```
      7/ TAKE 1, 641'      ; Puts a copy of pointer to ':='
      8/ DECR 0, 1         ; marker in acc. 1.
309   CODE: 9/ SKAN 1, 641' ; end of second half (compiling
     10/ JUMP * FIN1       ; code on left of ':=')
     11/ TAKE 2,*-641'     ; look at next character.
     12/ JAT 2, 23/        ; I or F-word (identifier).
     13/ SKET 2, 1         ; 1 contains a P-word.
     14/ JUMP 17/          ; S-word (extra code etc.).
     15/ OR 2, +6          ; change instruction to F 6,6.
     16/ JUMP 20/          ; jumps to a point of entry for
                           ; OINSTR.
     17/ SKAN 2, (seeeeeeeel); end of first half (compiling
     18/ JUMP * SEMIFIN1   ; code on right of ':=')
     19/ PTYP 5, 2         ; change type to a P-word.
     20/ TAKE 3, 2         ; OINSTR requires instruction to
                           ; be in acc. 3.
     21/ JUMP 7, * OINSTR  ; routine to output instruction.
     22/ JUMP * CODE       ; repeat code generation.
     23/ SKET 62, +2       ; skip if 2 contains an I-word.
     24/ JUMP * FWORD      ; array variable.
     25/ TAKE 3, * 641'    ; look at next character.
     26/ SKET 3, 1         ; acc. 1 contains a P-word.
     27/ JUMP 30/          ; not followed by a P-word
     28/ OR 3, 2           ; generates instruction F 6,A
     29/ JUMP 21/          ; point of entry for OINSTR.
     30/ DECR 0, 641'      ; move pointer back.
     31/ TAKE 3, [TAKE 6,0]; generates instruction
     32/ OR 3, 2           ;    TAKE 6, A.
```

Number Question Y must be in in this in

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

```
        33/ JUMP   21/                ; point of entry for OINSTR.
FWORD : 34/ TAKE   3, [INT 6, ]       ; instruction to ensure array
                                      ; subscript is an integer.
        35/ JUMP   7, *OINSTR         ; output instruction
        36/ TAKE   3, *641'           ;} is next character a F-word?
        37/ SKET   3, 1               ;}
        38/ JUMP   42/                ; not followed by a F-word.
        39/ OR     2, (P00064000);    changes 2 to P*A(6)
        40/ OR     3, 2               ; changes 3 to F 6,*A(6)
        41/ JUMP   21/                ; point of entry for OINSTR.
            DECR   0, 641'
        42/ TAKE   3, [TAKE 6,*0(6)]; generates instruction
        43/ OR     3, 2               ;}   TAKE 6,*A(6).
        44/ JUMP   21/                ; point of entry for OINSTR.
SEMIFIN1: 45/ TAKE 6, [MOCKS 645];    Resets pointer to compile
        46/ PUT    6, 641'            ;} second half.
        47/ JUMP   *CODE              ; repeat (only if destination is
                                      ; an array)
FIN1 : 48/ TAKE    2, 644'            ; look at destination.
        49/ SKET   5, 2               ; skip if 2 contains an I-word.
        50/ JUMP   1, 54/             ; F-word ⇒ destination is an array
        51/ TAKE   3, [PUT 6,0]       ;} compiles last instruction
        52/ OR     3, 2               ;}   PUT 6,A.
        53/ JUMP   1, 58/             ; to output last instruction.
        54/ TAKE   3, [INT 6, ]       ;} instruction to ensure subscript
        55/ JUMP   7, *OINSTR         ;} is an integer.
        56/ TAKE   3, [PUT 6,*0(6)];  compiles last instruction:-
        57/ OR     3, 2               ;}   PUT 6,*A(6)
        58/ JUMP   7, *OINSTR         ; output last instruction
```

Number of Sheet 46

Index Number

Name

The name need not be written on any sheet after the first

Number Question Y must be in in this in

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

```
59/ JUMP 1,*LEAVE        ; end of routine for ASSIGN
                         ; command.

OINSTR : INCR 0, 8       ; user page 1.
1/ TAKE 6,*[MOCKP|*10'(1)]; look at address in protection
                         ; register.
2/ SUBT 6,*[MOCKP 1,*4(1)]; look at address of next free
                         ; cell (copy of ASP in 4).
3/ JLZ 6, ERRORL         ; overflow.
4/ PUT 3,*4              ; put instruction into next free
                         ; cell.
5/ INCR 0, 4             ; increase ASP.
6/ PUT 0, 8              ; change back to user 0.
7/ PUT 6, 0              ; unload test of overflow from
                         ; stack since JLZ 6, does not
                         ; pop.
8/ JUMP *6               ; return ('pops' return address).

ERRORB : $CAPN           ;
1/ "WRON"                ;
2/ "G NU"                ; Prints out message:
3/ "MBER"                ; WRONG NUMBER OF BRACKETS.
4/ " OF "                ;
5/ "BRAC"                ;
6/ "KETS"                ;
7/ LINE                  ; outputs a new line.
8/ JUMP * LEAVE          ; end of routine for ASSIGN.
```

Number Question must be in this

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

```
ERROR : CAPN        ;
        1/ "ERRO"   ; Prints out message:
        2/ "R AT"   ;   ERROR AT :
        3/ ": &&"   ;
        4/ PUT Ø,8  ; user page Ø.
        5/ TAKE 2,+6  ; counter
        6/ TAKE 1,*1'  ; look at next character
        7/ SKAN 1,+62  ; newline?
        8/ JUMP 12/  ; leave routine
        9/ TOUT 1  ; output character
        10/ DECR Ø,2  ; decrease counter
        11/ JNZ 2,6/  ; repeat if six characters not
                      ; output.
        12/ LINE
        13/ JUMP * LEAVE  ; end of routine for ASSIGN.

ERRORL : CAPN       ;
        1/ "SPAC"   ;
        2/ "E OV"   ; Prints out message:
        3/ "ERFL"   ;   SPACE OVERFLOW
        4/ "OW&&"   ;
        5/ LINE
        6/ PUT Ø,8  ; user page Ø
        7/ TAKE 1,11'  ; copy of ASP at entering ASSIGN
                       ; routine
        8/ INCR Ø,8  ; user page 1
        9/ SKAN 1,4  ; skip if code not cleared
        10/ JUMP *LEAVEL  ; end of ASSIGN routine.
```

Number Question Y must be n in this n

Write on BOTH sides of the paper within the ruled margins. Do not spread your work out unnecessarily. This paper is not to be used as scribbling paper.

Leave this margin blank

```
        11/ PUT Ø, Ø'(1)        ; clear out generated code
        12/ INCR Ø, 1           ; increase pointer
        13/ JUMP 8/             ; to clear out next instruction

ERRORC : CAPN                   ;
         1/ "WRON"              ;
         2/ "G NU"              ; Prints out message :
         3/ "MBER"              ; WRONG NUMBER OF ':='
         4/ " OF "              ;
         5/ "':='"              ;
         6/ LINE
         7/ JUMP * LEAVE        ; end of routine for ASSIGN

LEAVE  : PUT Ø, 8               ; user page Ø.
         1/ PUT 4, 11'          ; replace assembly stage pointer
LEAVE1 : 2/ TAKE 1, 895'        ; return address.
           OR  1, (IØØØØ2ØØØ);
         3/ ~~TAKE 2, ASS~~     ;
            TAKE 2, 642'        ; points to assembler page
         4/ ~~PUT 2, 8~~        ;
            PUT 2, 6'
         5/ ~~OR 1, (IØØØØ2ØØØ)~~ ; to change to page bit set
            TAKE 2, ASS
         6/ ~~TAKE 2, 642'~~    ;
            PUT 2, 8            ; empty stack
         7/ ~~PUT 2, 6'~~       ;
         8/ JUMP *1             ; jump back into assembler to
                                ; list code
```