CONTENTS

Introduction to BBCX

Key to terms used 1

Hardware - allocation of user pages 1

Type bits 2

Instruction word format 2

The functions 3

Accumulators, index registers, indirect bit 3 - 7

Page bit 8

Combination of index indirect and page bit 8

I-words and S-words 9

Character set 10

F-words 11

Bytes and MOCKP words 12 - 13

MOCKS words 13 - 15

The hardware stack 15

The special properties of page Ø 16

Extracodes - complete table 17 - 21

READ 21 - 22

PRINT 23

PAGE 24

The Assembler - general description 25

P-words - general format 26

Identifiers 26

Literals 27

Extracodes and labels 27

F-, I- and S-words 28

Limitations 28 - 29

The Command language - BREAK, EDIT 29

RUN, CONTINUE, TAPE, KEY, ADVANCE, NEW, OLD, RENAME 29

SAVE, REPLACE, UNSAVE, LOGOUT, TRACE, CHECKA, CHECKN 30

REMOVE, LIST, DUMP, DECLARE, PLACES, ASSIGN 30

Example of ASSIGN 31

Array variables 32

IF (with example) 33

LOOP, REPEAT 34

Example of LOOP 34 - 35

REPEAT 35

LOOP variants 36 - 37

MESSAGE 37

38

BBCX

This is a preliminary specification of BBCX - a version of BBC currently being implemented on the PDP-10 at the Hatfield Polytechnic.

BBCX, like its predecessors, is the simulation of a computing system designed as a teaching tool to explain to the beginner some of the facets of computer design and organisation. In particular, through its study, it is hoped that students will learn to appreciate the distinction between hardware and software and it is for this reason that BBCX is presented as a complete machine together with its associated software, not simply as a computer language. The BBCX machine is a multiaccess computer and it is hoped that students will learn to understand some of the problems of timesharing and appreciate the particular solutions to these problems that have been used in the BBC design.

The beginner is shown initially only part of the total machine and because the design has been made with this in mind, topics can be introduced in an order dictated solely by educational requirements.

The timescale for the project is as follows:

September 1970 - Formal definition complete. Implementation commenced by the Computer Centre.

January 1971 - This specification is circulated to potential users. In a modified form, this specification is to be incorporated as an appendix to a full teaching manual along the lines of the BBC3 manual.

April 1971 - A version of BBCX becomes available for use on a limited scale. The full manual becomes available in draft form.

September 1971 - Completion date.

W. TAGG
December 1970.

Key to terms used

MSE most significant end (ie. the left end of a store content)
LSE least significant end (ie. the right end of a store content)
A The address of the accumulator specified by an instruction
a The content of A
N The address of the store operand specified by an instruction
n The content of N
r The result when a function has been obeyed
C( ) Content of

THE HARDWARE

The store consists of a number of "pages" and each page consists of 1024 cells (or store locations). A particular user has nearly complete control of up to 4 pages which for him are numbered 0, 1, 2 and 3. He can use these for program or for data but he also has access to pages with negative page numbers on a shared basis with other users. These shared pages contain permanent software which the user needs when he assembles, modifies or lists his program (or data) and because they are being shared they are available only for use (or to be read) and cannot be altered by a user.

Each page 0 has special properties and it is only later that the user will become familiar with its precise purpose. Briefly, it contains such things as characters input to the computer but not yet processed by it, characters which are queuing up for output and workspace associated with his use of the negative pages.

Page 1 is his main page and it is this page that a user automatically gains access to when he types .R BBCX followed by a response to NEW or OLD--. Initially it is wiped clear - ie. all the cells are filled with zeros.

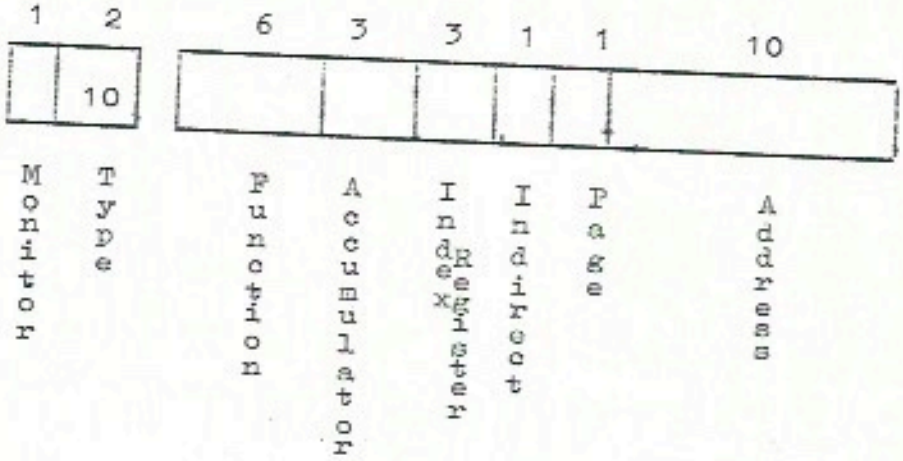
The first 8 cells of each page (those with addresses 0 - 7) have special properties but apart from these properties all 1024 cells of a page are identical. Each is capable of storing a small piece of information suitably coded into binary. Normally this information can be thought to belong to one of four types and the design of the computer (unlike most computers) makes special provision for these by allocating 2 bits to specify between them the following four alternatives:

00	I-word.	When the type bits are both set to zero the rest of the bits are to be interpreted as a positive or negative integer.
01	F-word.	01 specifies a number stored in floating point form. This is the method used to represent and store numbers which have a fractional part.
10	P-word.	10 specifies that the rest of the bits are to be interpreted as a 'program word' or instruction. Only words of this type are meaningful to the computer in the sense that only these can cause the hardware of the machine to act constructively.
11	S-word.	An S-word, specified by two ones, is a group of up to 4 characters packed together within the one cell.

Apart from the two type bits, each cell is divided into one monitor bit and 24 information bits. The monitor bit is not manipulated directly by the user, it is there to make sure that the user does not overwrite the content of any important

cell which the system is using to organise the timesharing properties of the machine. Sometimes, the monitor bit is used to provide runtime monitoring for the user but most of the time the user can forget its existence.

Instruction (P-word)



Function

The following table describes the action of the computer when it obeys one of the 64 different functions. The mnemonic associated with each function is the name used by the programmer when he is using the standard assembler, this name has nothing to do with this definition of the BBC hardware but is included for the convenience of the reader.

FUNCTION TABLE (ORDER CODE)

Binary Value	Mnemonic (alternative)	EBCD equivalent	Legal Acc operand I F P S	Legal Store operand I F P S	Description (Acc operand = a, Store operand = n, Result = r)	
000 000	NIL (NTHG)	NTHG	not used	not used	Do nothing	
000 001	OR	OR	Y Y Y Y	Y Y Y Y	Logical inclusive OR between bits of a and n. r replaces a.	The type bits are not affected. type (r) = type (a)
000 010	NEQV	NEQV	Y Y Y Y	Y Y Y Y	Logical non-equivalence (exclusive OR) between bits of a and n. r replaces a.	
000 011	AND	AND	Y Y Y Y	Y Y Y Y	Logical AND between bits of a and n. r replaces a.	
000 100	ADD	ADD	Y Y N N	Y Y N N	r=a+n and replaces a	r is an F-word unless i) a and n are both I-words and (in the case of division) ii) n divides exactly into a
000 101	SUBT	SUBT	Y Y N N	Y Y N N	r=a-n and replaces a	
000 110	MULT (MPLY)	MPLY	Y Y N N	Y Y N N	r=a*n and replaces a	
000 111	DVD		Y Y N N	Y Y N N	r=a/n and replaces a	
001 000	TAKE	TAKE	not used	Y Y Y Y	r=n and replaces a (ie. copy from store to Acc)	
001 001	TSTR		not used	Y N N N	take from store and stretch to form double length result. LSE replaces ACC A as specified and MSE (all '0's for positives n and all '1's for negative n) replaces Acc A-1. Both sign bits are correctly set.	
001 010	TNEG	NEG	not used	Y Y N N	r=-n and replaces a. type (r) = type (n)	
001 011	TNOT	NOT	not used	Y Y Y Y	r if formed by replacing '0's for '1's and '1's for '0's in n. type (r) = type (n). r replaces a.	
001 100	TTYP	TYPE	not used	Y Y Y Y	r = type (n) and replaces a (ie. the result is an I-word from 0 to 3 which specifies the type of n).	
001 101	TTYZ		not used	Y Y Y Y	r is an I-word whose bits are copies of the bits of n	
001 110	TTTT				not used at present	
001 111	TOUT	OPUT	not used	Y N N Y	Output the character defined by the 6 LS bits of n	

FUNCTION TABLE (ORDER CODE)

Binary Value	Mnemonic (alternative)	BBC3 equivalent	Legal Acc operand I F P S	Legal Store operand I F P S	Description (Acc operand = a, Store operand = n, Result = r)		
010 000	SKIP		not used	not used	Miss out the next instruction (ie. increase SCR by 2 instead of 1)		
010 001	SKAE	SKAE	Y Y Y Y	Y Y Y Y	Skip if a=n and type (a) = type (n)		
010 010	SKAN	SKAN	Y Y Y Y	Y Y Y Y	Skip if a≠n or type (a) ≠ type (n)		
010 011	SKET	SKET	Y Y Y Y	Y Y Y Y	Skip if type (a) = type (n)		
010 100	SKAL	SKAL	Y Y N N	Y Y N N	Skip if a < n	a and n do not need to be of equal type for a skip to take place.	
010 101	SKAG	SKAG	Y Y N N	Y Y N N	Skip if a > n		
010 110	SKED		Y Y N N	Y Y N N	Skip if a = n otherwise replace a by a-1		
010 111	SKEI		Y Y N N	Y Y N N	Skip if a = n otherwise replace a by a+1		
011 000	SHL		Y Y Y Y	Y N N N	Left logical shift of n places on a	n must satisfy $-24 \leq n \leq 24$	The type bits of the accumulator are not affected.
011 001	ROT		Y Y Y Y	Y N N N	Left cyclic shift of n places on a		
011 010	DSHL		Y Y Y Y	Y N N N	Left logical shift on double length acc operand specified by A and A-1. A is LSE of the operand and A-1 the MSE.	n must satisfy $-48 \leq n \leq 48$	
011 011	DROT		Y Y Y Y	Y N N N	Left cyclic shift of n places on double length acc operand specified by A and A-1. A is LSE of the operand and A-1 the MSE.		
011 100	POWR		Y Y N N	Y Y N N	r = a↑n and replaces a. The result is an F-word except when n=0 or when a and n are I-words and the result is an integer. When a < 0 and n is an F-word, r is undefined and an error interrupt occurs.		
011 101	DMULT		Y N N N	Y N N N	Double length multiplication using 47 bit acc operand and 24 bit store operand. Result replaces acc operand which is formed from A(LSE) and A-1 (MSE). Both sign bits are correctly set.		
011 110	DIV		Y N N N	Y N N N	Single length integer division with remainder left in ACC A-1 Rem always > 0		
011 111	DDIV		Y N N N	Y N N N	Double length division using double length dividend and quotient and single length divisor. Quotient calculated on assumption that remainder is positive. However the remainder is not kept. Acc A and A-1 are used as in DMULT.		

Binary Value	Mnemonic (alternative)	BBC3 equivalent	Legal Acc operand I F P S	Legal Store operand I F P S	Description (Acc Operand = a Store operand = n, Result = r)
100 000	HLX (SWAP)	XITHG	Y Y Y Y	Y Y Y Y	These functions are identical to those with binary values 000 000 to 000 111 except that finally the content of the accumulator and the specified store location are exchanged.
100 001	ORX	XOR	Y Y Y Y	Y Y Y Y	
100 010	NEQVX	XNEQV	Y Y Y Y	Y Y Y Y	
100 011	ANDX	XAND	Y Y Y Y	Y Y Y Y	
100 100	ADDX	XADD	Y Y N N	Y Y N N	
100 101	SUBTX	XSUBT	Y Y N N	Y Y N N	
100 110	MULTX (MPLYX)	XMPLY	Y Y N N	Y Y N N	
100 111	DVDX		Y Y N N	Y Y N N	
101 000	PUT	PUT	Y Y Y Y	not used	r = a and replaces n (ie. copy from Acc to Store)
101 001	PSQU		Y N N N	not used	Put squashed. Convert a double length acc operand stored in A and A-1 to a single length result to replace n. Single length overflow occurs unless the bits of A-1 are copies of the sign bit of A.
101 010	PNEG		Y Y N N	not used	r = -a and replaces n. type (r) = type (a)
101 011	PNOT		Y Y Y Y	not used	r is formed by placing '0's for '1's and '1's for '0's in a. type (r) = type (a). r replaces n.
101 100	PTYP	CHYP	Y N N N	not used	the type bits of n are replaced by new type bits formed from the two LS bits of a. The rest of n is unchanged.
101 101	PTYZ		Y Y Y Y	not used	r is an I-word whose 24 information bits are formed by copying the 24 information bits of a. r replaces n.
101 110	PPPP				not used at present.
101 111	PIN	IPUT	not used	not used	r is an S-word with next character from the input buffer at its LSB. If the buffer is empty, DATA* is output and the program is suspended until a new line of input has been typed.

Binary Value	Mnemonic (alternative)	BBC3 equivalent	Legal Acc operand I F P S	Legal Store operand I F P S	Description (Acc operand = a, Store operand = n, Result = r)
110 000	JUMP		not used	N N Y N	SCR is changed to N so that the instruction to be obeyed next is the instruction in N. The old value of SCR (the addr of next instr) is first stored in Acc A-1
110 001	JEZ		Y Y Y Y	N N Y N	Jump if a = 0. The type bits of a are not inspected.
110 010	JNZ		Y Y Y Y	N N Y N	Jump if a ≠ 0. The type bits of a are not inspected.
110 011	JAT		Y Y Y Y	N N Y N	Jump if type (a) is arithmetic. (ie. if a is an I- or F-word.)
110 100	JLZ	JLZ	Y Y N N	N N Y N	Jump if a < 0.
110 101	JGZ	JGZ	Y Y N N	N N Y N	Jump if a > 0.
110 110	JZD		Y Y N N	N N Y N	Jump if a = 0 otherwise replace a by a-1
110 111	JZI		Y Y N N	N N Y N	Jump if a = 0 otherwise replace a by a+1
111 000	DECR		i) not used ii) not used	Y Y N N N N Y N	r=n-1. r replaces a and n. r=P-word with addr field reduced by 1* (0 decreases to 1023) r replaces a and n
111 001	INCR		i) not used ii) not used	Y Y N N N N Y N	r=n+1. r replaces a and n. r=P-word with addr field increased by 1* (1023 increase to 0). r replaces a and n.
111 010	MOCKP		N N N N	N N N N	Illegal as an instruction. Used only as an operand for indirect addressing.
111 011	MOCKS		N N N N	N N N N	
111 100	DPME		Y Y Y Y	Y Y Y Y	The store operand must be a byte accessed by a MOCKP or MOCKS pointer word. This byte is not used but its place is taken by a copy of the appropriate number of bits from the LSE of the accumulator. When MOCKS is used and the byte is at MSE of the word, the type bits of the previous word are used to overwrite the type bits of the current word.
111 101					
111 110	EXEC	EXEC	as appropriate	N N Y N	Obey, out of sequence, the instruction pointed to by N. The instruction may not be another EXEC instruction.
111 111	EXTRA	LIBR	as appropriate	- - - -	The address of the store operand is not used to specify an operand. Instead it is used to select an extracode (ie. a piece of software within the monitor to perform for the user a task which although often required is not available as a hardware function). All the extracodes have separate alternative mnemonics.

*If the P-word is a byte pointer, INCR and DECR alter the P-word so that it points to the next byte instead of the next whole word.

Accumulator

Any of the first 8 cells (ie. with addresses from 0 to 7) can be specified as an accumulator. Accumulators 0, 6 and 7 have special properties but the others are identical.

Special properties:

- 0 Always appears as the source of zero (I-word) any value assigned to 0 is lost.
- 6 This is the top of a hardware stack with a "last in first out" property. See separate section on hardware stack.
- 7 This is the sequence control register (SCR). It contains as a MOCKP word a pointer to the next instruction to be obeyed.

Index register

Any of the first 8 cells (ie. with addresses 0 to 7) can be specified as an index register. When an index other than 0 is used it causes modification of the final address of the store operand. This address is modified by the addition of the content of the index register. Because it is only legal for a register used as an index to contain an I-word or an S-word, the use of cell 7 as an index always causes an error.

Indirect bit

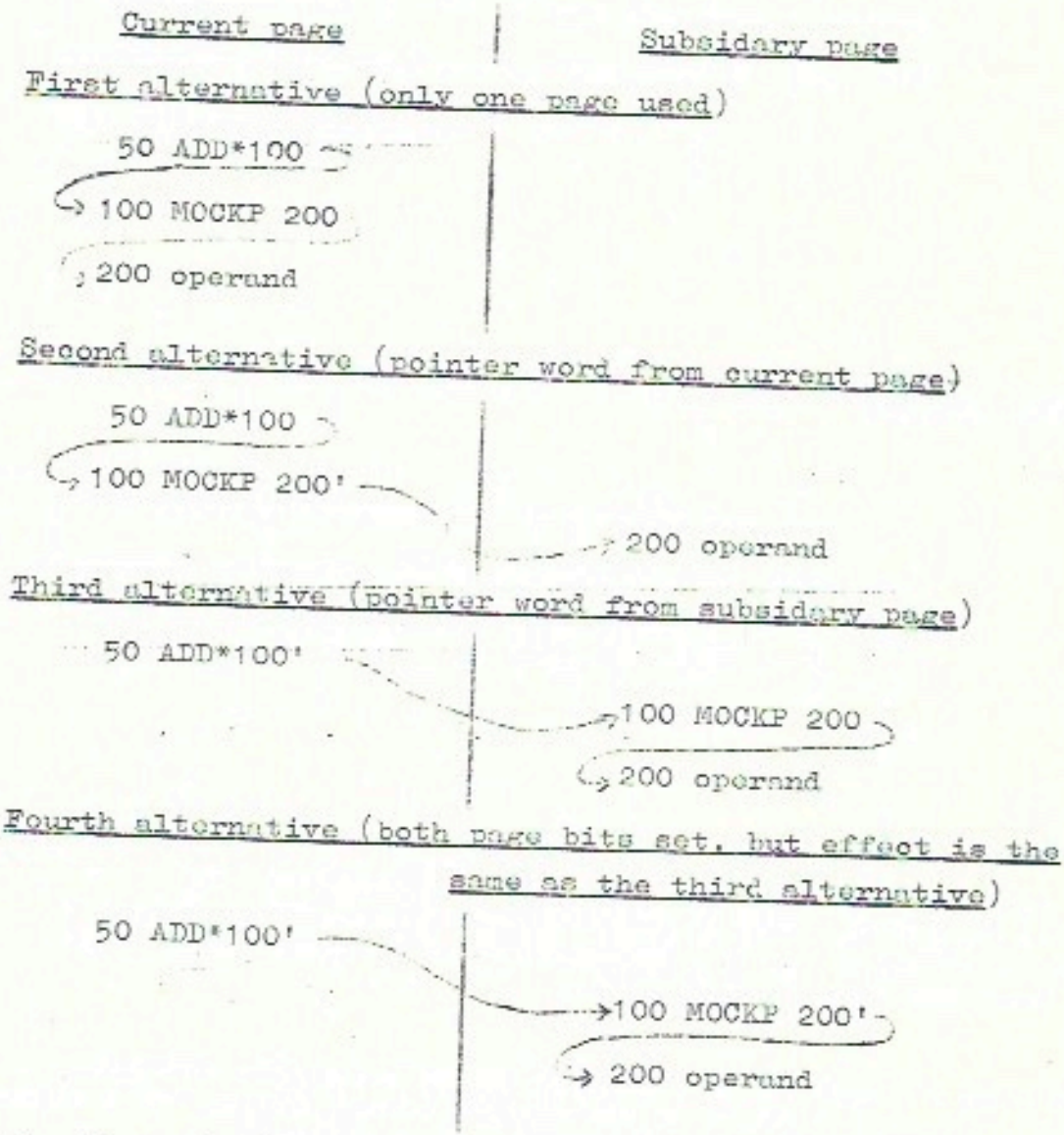
Before a store address is modified by an index, the indirect bit is inspected. If it is zero the address part of the word is used directly but if it is a one, a second word is retrieved from store using the address field of the first word. This second word must also be a P-word for its address field is used to retrieve the store operand. It is the address of this second word which is modified (if necessary) by the content of the index register specified in the first instruction. (The rest of the bits of the second word are normally only used under special circumstances concerned with retrieving and depositing bytes.)

Page bit

A page bit is associated with each store access. If the page bit is zero the corresponding address specifies a cell within the current page, ie. the same page as the current instruction itself. If the page bit is a one, then another page is referred to. The user cannot specify absolutely which this other page is to be for the responsibility of allocating pages to users is a monitor responsibility. Setting the page bit to a one forces control to pass to the monitor so that the monitor can inspect the content of location 8 of the user's current page to determine which store content it is to hand over. Thus it is necessary for the user to make sure that location 8 contains the correct information before the instruction with the page bit set is obeyed.

A user specifies another page either by loading an I-word into 8 (an integer from 0 to 3 to specify the page number) or by loading an S-word into 8 (specifying the page name).

When indirect addressing is used, the second or both store accesses might be from another page. When the second store access is about to be made, the page bit of the pointer word is only inspected if the pointer word itself was retrieved from the current page. Thus store assess never involves three pages and the following alternatives, can occur:

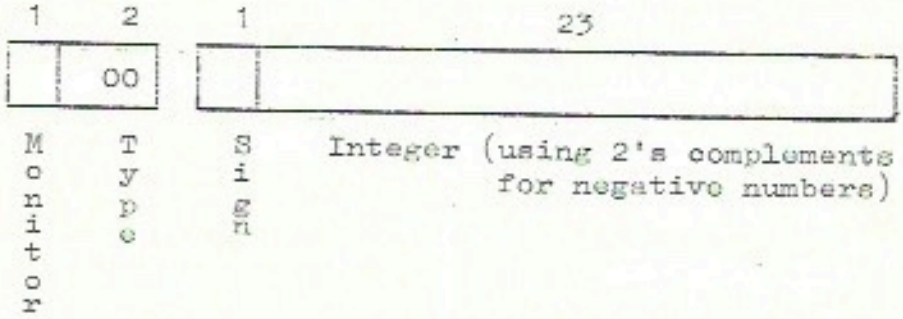


NB. The asterisk causes the indirect bit to be set and the apostrophe causes the page bit to be set.

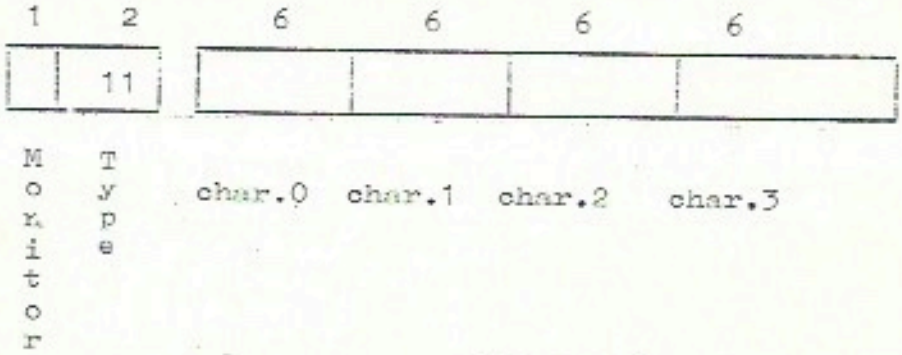
When a jump instruction references an operand in another page the effect is to change the current page. In this case, it is the SCR of the new page which is set by the jump instruction and the SCR of the old page is left pointing to the instruction in the old page that would

have been obeyed next if the jump had not taken place. The monitor loads location 8 of the new page with the name of the old page so that an exit back to the old page is easily achieved with the instruction JUMP *7'. A jump to page zero is illegal.

Integer (I-word)



String (s-word).



Up to 4 characters can be stored in packed form in a given location. The character representation is given in the table belpw and is independent of any code that might be used for tape preparation.

The null character can be typed in by using the ampersand (&). On output it does not appear at all except when quotes are used when again it appears as an ampersand.

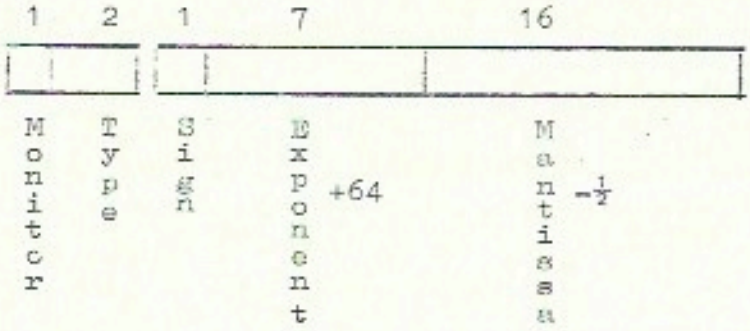
CHARACTER SET

Decimal	Octal	Binary	Character	Decimal	Octal	Binary	Character	Decimal	Octal	Binary	Character	Decimal	Octal	Binary	Character
0	00	000 000	null	16	20	010 000	P	32	40	100 000	Ø	48	60	110 000	⊓
1	01	000 001	A	17	21	010 001	Q	33	41	100 001	1	49	61	110 001	⊓
2	02	000 010	B	18	22	010 010	R	34	42	100 010	2	50	62	110 010	*
3	03	000 011	C	19	23	010 011	S	35	43	100 011	3	51	63	110 011	/
4	04	000 100	D	20	24	010 100	T	36	44	100 100	4	52	64	110 100	=
5	05	000 101	E	21	25	010 101	U	37	45	100 101	5	53	65	110 101	≠
6	06	000 110	F	22	26	010 110	V	38	46	100 110	6	54	66	110 110	←
7	07	000 111	G	23	27	010 111	W	39	47	100 111	7	55	67	110 111	↑
8	10	001 000	H	24	30	011 000	X	40	50	101 000	8	56	70	111 000	?
9	11	001 001	I	25	31	011 001	Y	41	51	101 001	9	57	71	111 001	"(quote)
10	12	001 010	J	26	32	011 010	Z	42	52	101 010	.	58	72	111 010	:
11	13	001 011	K	27	33	011 011	apostrophe	43	53	101 011	subscript 1 10	59	73	111 011	;
12	14	001 100	L	28	34	011 100	<	44	54	101 100	+	60	74	111 100	,
13	15	001 101	M	29	35	011 101	>	45	55	101 101	-	61	75	111 101	space
14	16	001 110	N	30	36	011 110	≤ 2	46	56	101 110	(	62	76	111 110	new line
15	17	001 111	O	31	37	011 111	≥ 2	47	57	101 111	)	63	77	111 111	altmode 3

1 @ is used.
2 a '<' or '>' sign is followed immediately by an '=' sign. The two characters are regarded by the computer as a single '≤' or '≥' sign.
3 Altmode. Echoes as a dollar and returns control to the BBC monitor.

Floating point (F-word)

Numbers other than integers are normally stored in floating point form. Although the method is complicated it allows all numbers between about $\pm 10^{19}$ to be stored to about 5 figures of accuracy.



The number is split first of all into 3 factors. The first factor (the sign) is either +1 or -1, the second factor is some exact power of 2 (positive or negative) and the third factor (the mantissa) is a fraction $\geq \frac{1}{2}$ and < 1. The mantissa is now expressed as a binary fraction and rounded to 17 binary places. The most significant of these (just to the right of the binary point) is discarded for since the mantissa is at least $\frac{1}{2}$, this bit must always be a one so that it can always be reconstituted when the number is retrieved from store. The 16 remaining bits are stored at the LSB of the word.

The two's exponent of the second factor is now dealt with. If the original number is within range, this exponent will be an integer ≥ -64 and $\leq +63$. 64 is added to the exponent so that the result is never negative and the result expressed as a 7 bit integer, is packed into the word with the mantissa.

Now the sign of the number is dealt with. If positive, the sign bit is left zero and the number is complete, but if it is negative then the whole word is 2's complemented leaving the sign bit a one.

Example To express -3.25 in floating point form

$-3.25 = -1 * 2^2 * 13/16$ ... not stored

mantissa $= 13/16 = (.11010000000000000)_2$

exponent + 64 = 2+64 = 66 = $(1000010)_2$

full word before two's complementation

0 1000010 1010000000000000

after two's complementation

1 0111101 0110000000000000

Using the above conventions presents two difficulties

First difficulty

The F-word 0 0000000 0000000000000000

should represent $2^{-64} \times \frac{1}{2}$ but the effect of two's complementation on this leaves it unchanged so for this reason

$2^{-64} \times \frac{1}{2}$ is not representable and if this is the true result of a calculation the result is stored as the I-word zero.

Second difficulty

The F-word 1 0000000 0000000000000000

should not appear at all since this too is the two's complement of itself.

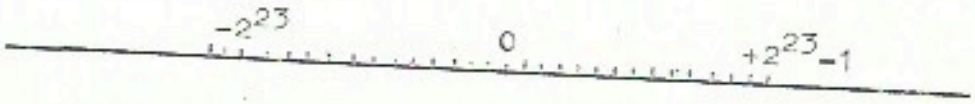
The intricacies of floating-point storage are all coped with my the hardware of the machine so that the user need not sorry too much about the details, it is sufficient if he is aware of the following:

i) Most machines employ both floating point and integer arithmetic although the details of the floating point storage vary from one machine to another. However unlike the BBCX machine, most computers use separate machine functions for manipulating numbers stored in floating point form.

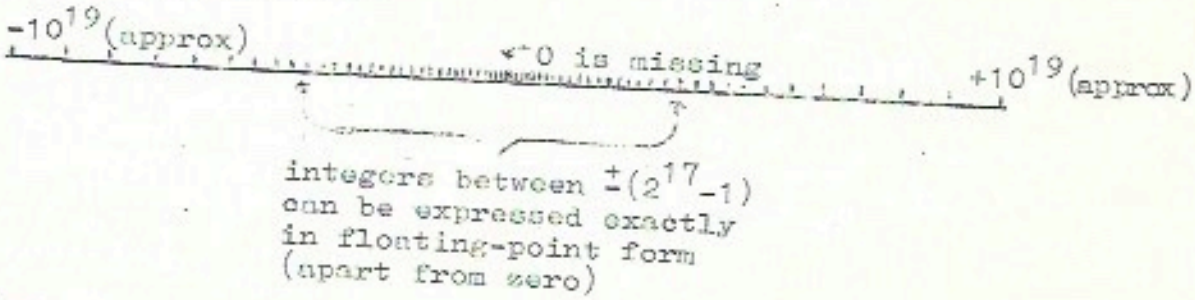
ii) Floating-point arithmetic involves the manipulation of binary patterns within the machine which may not exactly represent the corresponding numbers. In a binary machine, even apparently harmless numbers like 3.6 cannot be stored exactly since .6 is a recurring binary fraction.

iii) No matter how numbers are represented, 24 bits cannot be used to represent more than 224 alternatives. Floating-point form allows for numbers which can be represented exactly to be densely packed if they are numerically small and sparsely if they are numerically large. This is shown diagrammatically below :

I-word representation:



F-word representation:



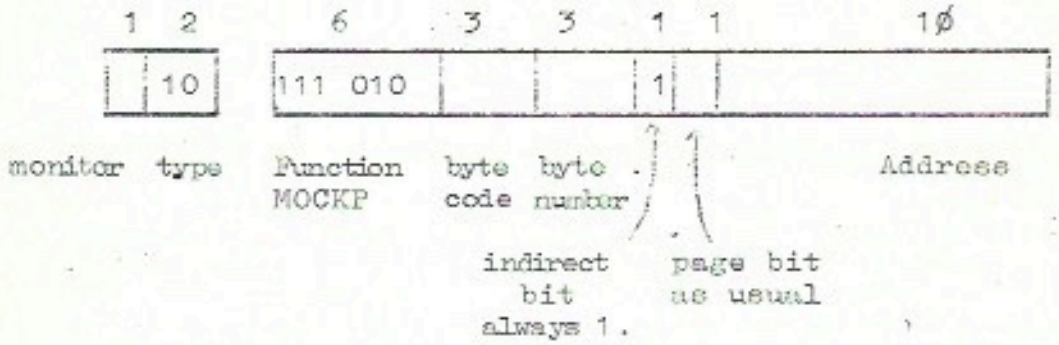
Bytes

Within the instruction execution cycle, the computer selects operands from 1 or more accumulators and from a location in store and delivers a result to any of those places. Apart from the store operand, all the operands are full length and the result is also full length.

Under normal circumstances, the store operand is a complete word of 27 bits (24 information bits, two type bits and 1 monitor bit) but when indirect addressing is used it is possible for this operand to be a byte of 24, 12, 8, 6, 4, 3, 2 or 1 bits. When this is the case the byte is regarded as the lower part of an I-word and the more significant bits are set to zero so that a full length word is handed over to the function.

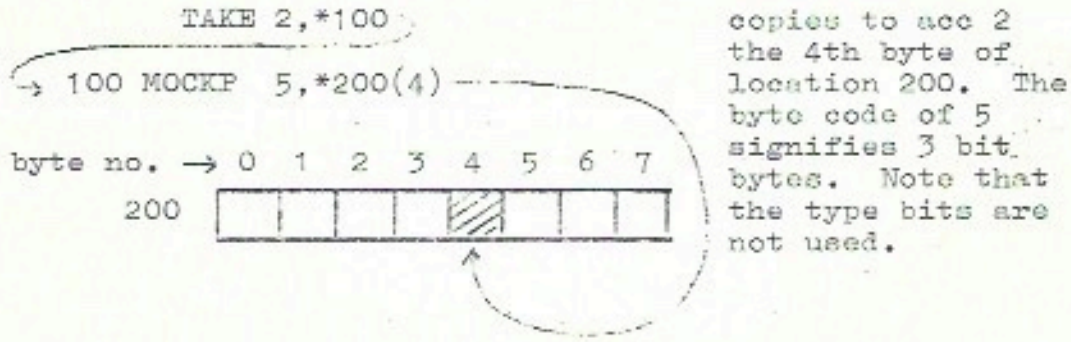
Indirect addressing demands that the content of the pointer word should itself be a P-word. This P-word can be another genuine instruction, a NIL instruction (as used by the assembler when labels and indirect jumps are used) or a mock instruction (MOCKP or MOCKS). It is only in the case of MOCKP and MOCKS P-words and when the indirect bit of these words is set that a store operand is a byte rather than a full word. Pointer words of this format are recognised during the instruction execution cycle and are dealt with exceptionally.

MOCKP



When MOCKP is used the bits within the accumulator field are used to define the byte code and the bits within the index field the byte number.

eg:



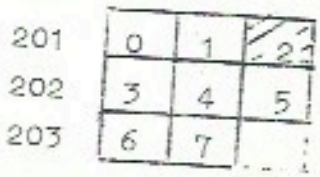
The byte code specifies the byte size according to the following table:

Byte Code	No. of bytes per word	No. of bits per byte
0	1	24
1	2	12
2	3	8
3	4	6
4	6	4
5	8	3
6	12*	2
7	24*	1

*only the first eight are accessible since the byte number cannot exceed 7.

When the byte code is less than 5 there may be more than one way of referring to the same byte. eg:

MOCKP 2,*200(5) and MOCKP 2,*201(2) both refer to the 8 least significant bits of location 201.



MOCKS

When MOCKS is used as a pointer the result is the same as it is when MOCKP is used. However, as a separate operation, the MOCKS P-word itself is modified so that it points to the next byte or whole word.

MOCKS pointer with indirect bit unset

In this case the pointer is accessing whole words so that the address field of the MOCKS word is incremented after the rest of the instruction is complete.

MOCKS pointer with indirect bit set

First of all the byte number is incremented in such a way that overflow from the index field (specifying the byte number) causes the address field to be incremented. Secondly, the byte code is inspected and if the new byte number exceeds the byte code it is reduced by one more than the byte code and the address field is incremented. In this way, the MOCKS P-word always addresses a byte in a standardised manner.

The Hardware Stack

In each page, accumulator 6 provides direct access to the stack in page 0. Normally the use of accumulator 6 provides access to the top element of the stack so that accumulator 6 behaves like any other accumulator but there are certain exceptions:

1) Any instruction in the TAKE group (001 000 to 001 111) used with accumulator 6 "pushes down" the previous value of acc 6 and assigns a new value to the top of the stack.

2) Any instruction in the PUT group (101 000 to 101 111) used with accumulator 6 assigns the top of the stack to store in the usual way but this value is then lost from the accumulator and its place is taken by the value underneath which "pops up".

3) Any instruction whatsoever which uses address 6 as the address of the store operand (either directly or indirectly) or as an index causes the used value to be removed from the stack.

4) JUMP 7,---- pushes down the top of the stack and sets the new top equal to the link address as a MOCKP word.

The use of the stack is illustrated in the sections dealing with the compiling commands (ASSIGN, IF and LOOP), the examples that follow are used to illustrate what typically happens in special circumstances.

		stack before	stack after
i)	TAKE 6,+2	5.3 6.2	2 5.3 6.2

Adds an item to the stack

ii)	PUT 6,100	2 5.3 6.2	5.3 6.2

The "2" is transfered into cell 100

iii)	ADD 6,+4	2 5.3 6.2	6 5.3 6.2

This instruction modifies the top item on the stack

iv)	SUBT 6,6	2 5.3 6.2	3.3 6.2

The store operand is 2 (this item is now removed) and the acc operand is 5.3. The result is left in place of 5.3.

v) ADD 6,100(6)

stack before	stack after
2	
5.3	9.6
6.2	6.2

The top item is removed and used as an index modifier and the next item is used as the acc operand. The result shown here assumes that C(102) = 4.3

vi) SKEI 6,6

2	
5.3	6.3
6.2	6.2

The top two items are compared (store operand = 2 and acc operand = 5.3). The top item is removed. If the two operands are equal, a skip takes place otherwise (as in this case) the acc operand is increased by 1.

vii) 100 JUMP 7,*L1

5.3	MOCKP 101
6.2	5.3
	6.2

This instruction causes a jump to the label L1 but only after a copy of the SCR (MOCKP 101) has been placed on the stack.

<u>The special properties of page Ø.</u>

Page Ø always has the name ZERO and is made available to the user as a page loaded in core as soon as the user gets into BBCX by typing .R BBCX. It has special hardware properties and areas within Page Ø are used as work space by the monitor, assembler and other pieces of the systems software. In fact it becomes the page Ø of the system software at times when this software is servicing the user. Because of its special nature it can never become the current page. It has no SCR and if it contains code at all this can only be copies of instructions to be obeyed elsewhere or instructions to be obeyed singly via the EXEC instruction.

Input and output take place autonomously without reference to the central processor and control certain switches which the monitor can test to see if a particular input or output transfer is complete. Input is always on a line-by-line basis except that a altmode is trapped (again by setting a switch). Output is normally on the basis of several characters at once rather than individual characters.

The individual cells within page Ø are used as follows:

0 : Contains a zero I-word as usual.

1 : The input pointer. Input to all programs from this teletype use this pointer. When a line of input is complete (ie. when the carriage return is typed) the user status (location 5 of page Ø) is adjusted and C(1) is set to:

MOCKS 3,*384

so that the first byte is retrieved from the MSE of location 384 in page Ø.

2 : Not used.

3 : The output pointer. When a group of characters have been assembled for output, the characters are loaded into an output buffer in page Ø (by an extracode, a TOUT instruction or directly), the characters are terminated by a (non-printing altmode), the output pointer is set to:

MOCKS 3,*640

the user status is adjusted and the program is aborted until the output is complete. Termination of the output resets the user status.

4 : The user's clock. When this cell is set to a positive I-word, it is decremented each time a user instruction is obeyed and an interrupt occurs when it reaches zero. The clock is not altered if its content is anything other than a positive I-word.

5 : The user status. (Protected — ie. the monitor bit is set). This cell contains as an I-word a positive integer from 0 to 15. The least significant bit specifies input (0) or output (1) status, the next bit whether (0) or not (1) the current program is running, the next bit to distinguish between an input/output interrupt (1) and a clock interrupt (0) and the next bit specifies monitor (0) or user (1) mode. Under monitor mode, page Ø essentially belongs to the monitor and not to the user.

6 : Hardware stack pointer. The command RUN resets it to the value

MOCKP 128

so that the stack appears empty. Location 128 is also set to zero since this now becomes the top of the stack.

7 : The current page number (protected).

8 : The Base address for the assembler. The content of this cell is reset each time the command BREAK or EDIT is used. It contains as a MOCKP P-word the address which is to be used as a base for relative addressing during program assembly. EDIT sets the

base address to an I-word Ø and NEW sets it to MOCKP 16
BREAK sets it to a MOCKP word with an address part equal to the next address typed after BREAK.

9 : The Quotes marker. Set alternately to 1 and 0 by quotes signs when encountered by READ extra codes. RUN sets C(Quotes marker) = 0. See READ for further details.

10 : Places setting. Reset by the command PLACES and initially set to the I-word 5. See PRINT for further details.

11 : Assembler storage pointer. Although the user must give in absolute or relative form the address of each instruction he assembles, the assembler uses the storage pointer to:

i) Check that instructions are in sequence (in BREAK mode only)

ii) position pieces of automatically compiled code using ASSIGN IF LOOP REPEAT or MESSAGE.

Its value is up-dated after it has been used but left constant during EDIT.

12-15: not used.

16-19: Contain as S-words the names of the user's core pages. C(16) = "ZERO" (always), C(17) is the name of the user's main page (page 1) and C(18) and C(19) are the names of subsidiary pages if such pages exist and are I-word zeros otherwise.

All are protected (monitor bit set to 1) and can only be changed via the command language.

20-22: Contain the names of filed pages.

23-31: Protected. Contain a user's error statistics.

32-51: Normally, runtime errors are trapped and control passes to the monitor so that an error message can be output under the control of the systems software. However, the user has the option of servicing these errors himself. Each type of error is associated with a cell in page Ø. If the C(this cell) is zero, the servicing of errors of this type is the responsibility of the monitor but if the content is a P-word the instruction containing the error is removed from the current instruction register (without being obeyed) and the P-word from page Ø is copied into the current instruction register to be obeyed in its place.

32 : Square root of negative number.

33 : Log_e of negative or zero number.

34 : Read error during READ.

35 : Sin, cos or tan of an angle which is so large that significance is lost.

36 : PAGE error. Transfer to or from backing store of a non-existent page or to a non-existent destination.

37 : An attempt has been made to execute an EXEC instruction.

38 : The cell just accessed for the next instruction does not contain a P-word or contains a MOCKP or MOCKS word.

39 : Indirect addressing has been attempted with a pointer word which is not a P-word.

40 : Index modification has been attempted using a word that is neither an I-word nor an S-word.

41 : Operand of wrong type. The type of the acc operand or the type of the store operand or both are inconsistent with the function currently being obeyed.

42 : An attempt has been made (using either DVD or DIV or DDIV) to divide by zero.

43 : Integer overflow.

44 : Double length integer overflow.

45 : Floating point overflow.

46 : Shift store operand out of range.

47 : Operand sought from a non-existent page.

48 : Extracode address part out of range.

49 : An attempt has been made to assign a new value to a protected cell.

50 : Stack overflow. More than 128 items on the stack.

51 : Illegal F-word. No floating point number corresponds to an F-word whose 23 least significant bits are zero.

52 : Clock interrupt.

53 :

54-63: Not used.

64-68: Page Ø contains a copy of all system messages that can be output to a user. These can be changed if need be (but a user does so at his peril). Pointers to the messages are stored here.

64 : Points to the asterisk which is output inviting a user to type.

65 : Points to DATA* which is output inviting a user to type data to his own program.

66 : Points to the message END OF PROGRAM.

67 : Points to the message UNUSED DATA:

68 : Points to the message INTERRUPTED. L=

69 :

70-127: Not used

128-255: The hardware stack. The top element of the stack is accessed through location 6 in any page.

256-383: The system messages pointed to by the pointers in cells 64-68.

384-639: The input buffer area

640-895: The output buffer area

896-1023: Work space used by the systems software.

EXTRACODES

The extracodes are a set of routines within the monitor and performed by the monitor on behalf of a user. In general they take a single operand from an accumulator and leave a result in the same accumulator. A user incorporates them into his program either by using the mnemonic EXTRA followed by an appropriate accumulator and a pseudo-address to specify which extracode he requires or by using a special mnemonic.

Pseudo address	Mnemonic	Types of operand allowed	Result type	Remarks
1	SQRT	F or I	F or I	Square roots the Acc which must be ≥ 0. Result is an F-word unless the operand is an I-word and a perfect square.

Pseudo address	Mnemonic	Types of operand allowed	Result type	Remarks
2	LN	F or I	F	Natural log of Acc which must be > 0
3	EXP	F or I	F	Raises e to the power given by Acc. Floating point overflow occurs if Acc $\geqslant 63 \log_e 2$
4	READ	-	Any	See below
5	PRINT	Any	-	for details
6	SIN	F or I	F	The operand is in radians and, if an F-word, must be numerically less than 2^{17}. Floating-point overflow can occur when TAN is used.
7	COS	F or I	F	
8	TAN	F or I	F	
9	ATN	F or I	F	Arctan. The principal value is returned.
10	STOP	-	-	Outputs an END OF PROGRAM message and returns control to the BBC monitor.
11	LINE	-	-	Outputs a new line
12	INT	F(or I)	I	Finds the integer part, ie. the greatest integer not greater than the number.
13	FRAC	F	F	Finds the fractional part. The result is always positive so that x = INT(x)+FRAC(x) even when x is negative.
14	FLOAT	I	F	Changes the integer Acc content to floating-point form.
15	CAPN	-	-	S-words are placed in successive locations after CAPN. When the program is run, these S-words are printed out by the CAPN routine to form a caption. The instruction following the last S-word is obeyed next.
16	PAGE	I	-	See below for details
17	RND	I	I	A random integer from Ø up to but excluding α is returned
		F	F	A random F-word between 0 and α is returned.
18	ABS	I or F	I or F	Absolute value

READ

The READ routine is equivalent to a series of PIN instructions. The number of PIN instructions that are obeyed and the result that is returned to the specified accumulator depends on the characters that are read and on the current state of a cell within page Ø called the "quotes marker". When a user types RUN, the quotes marker is always set to zero and will only change from this value if a quote character (") is read or if the user deliberately changes it.

If the quotes marker is non-zero when READ is obeyed, characters are read and packed from left to right into the specified accumulator until either

i) 4 characters have been read or
ii) a new line character has been read or
iii) a quote is read. In this last case, the quote itself is not stored and the quotes marker is made equal to zero (I-word).

If the quotes marker is zero then the action taken by READ depends on the first significant character that is input. Spaces, commas and new lines are regarded as non-significant and are discarded and the rest of the characters are summarized below:

i) An alphabetic character. An S-word is formed from this and subsequent alphanumeric characters (ie. letters or digits). Reading stops as soon as a non-alphanumeric is met and the pointer to the input buffer is stepped back so that this character is the next to be input. If at this stage four or less alphanumerics have been read, then these are packed from left to right with any spares left as null characters. If more than four characters have been read, the first three and last are stored and the remainder are discarded.

ii) A digit, +, -, decimal point or subscript 10. A numeric value is expected. The final result will be an I-word if no decimal point or subscript 10 is read and an F-word otherwise. Reading normally ceases when some other character is read (with the input pointer again stepped back so that this character can be read again) but the terminating character can be a second decimal point or subscript 10, or exceptionally a + or - sign. The following examples clarify the position.

63.4.5 63.4 read, terminator : decimal point

$12.5_{10}+3+$ $12.5_{10}+3$ read, terminator : plus sign

iii) An altmode. The READ instruction is aborted, the SCR is restored to the value it had at the beginning of the instruction and control passes to the BBCX monitor.

iv) Space, comma and null character. Ignored initially but a space or comma can be used to terminate an I, F or S-word which is not in quotes. Nulls are ignored completely except in quotes.

v) A new line. Behaves like a comma or space except that when nothing significant has been read, a new line causes the current program to be temporarily aborted and DATA* is output on the terminal. When the user has typed in a line of characters the program is allowed to continue.

vi) An open bracket. Octal input is expected. This consists of an I, S, F or P immediately following the open bracket (to indicate the type of the word being read) followed by exactly 8 octal digits followed by a close bracket.

vii) Any other character. Causes a READ ERROR initially and can be used to terminate when an I, F or S-word not in quotes in read.

PRINT

The PRINT routine is equivalent to a series of TOUT instructions. The number of TOUT instructions that are obeyed depend on the content of the specified accumulator and on the current state of a cell within page Ø called the "places setting". The content of the places setting is an I-word and is initially +5. It can be changed by the command PLACES followed by a signed integer or by normal programming.

The effect of the PRINT routine is summarized in the following table:

Word type specified by acc.	State of "places setting"		
	PLACES+n	PLACES-n	PLACES+0
I-word	In decimal form to n places. Leading zeros are replaced by spaces and a space or - sign is printed. The number is terminated by two spaces so that n+3 characters are printed altogether Numbers that are numerically too large for this type of output are printed on a new line to 8 places.	Similar to PLACES+n except that: i) a + sign is printed in front of positive numbers in place of the space ii) Numbers that are numerically too large are printed to 8 places on the same line.	Output is in octal with format identical to the format required for octal input.
F-word	In decimal form to n places. Output starts with a space or - sign, followed by any digits before the decimal point followed by the decimal point itself and sufficient digits after the point to make n digits altogether. A single space terminates the output so that n+3 characters are printed altogether. Numbers that are too large are printed to 5 figures in decimal and exponent form on a new line.	Similar to PLACES+n except that: i) a + sign is printed in front of positive numbers in place of the space ii) Numbers less than 0.01 are printed in decimal and exponent form iii) Decimal and exponent form is not on a new line.	Octal output

Word type specified by acc.	State of "places setting"		
	PLACES+n	PLACES-n	PLACES+0
P-word	In mnemonic form but with a numeric address. Spaces terminate the output to make 15 characters altogether.	Identical to Places+n	Octal output
S-word	The characters comprising the S-word are output. Null characters are suppressed.	Exactly 4 characters are enclosed in quotes. Null characters appear as &.	Octal output

PAGE

Two page names are given as parameters in the next two cells. The first is the name of a page currently in core and the second is the name of a page from backing store. A copy of the content of one of these pages is used to overwrite the other (with the exception of the C(8) which is the page's name and any other word which has the ~~monitor bit set~~. The sign of the accumulator operand specifies the direction of transfer, + from core to backing store if positive and from backing store to core if negative.

A page of core must be defined via a ~~LOAD~~ NEW or OLD command before it can be overwritten by the content of a file. Thus a user's program cannot grow into new pages dynamically.

THE ASSEMBLER

The assembler is a program within a negative page which can be used to transform input data in the form of source program lines into their internal representation or object words. Only one copy of the assembler program is in core and this copy is run on behalf of each user in turn. For this reason this copy only consists of pure code (ie. it contains no variables) so that it is easy to ensure that a user leaves the assembler in a fit state for another user to take over. Approgram that has this capacity is said to be re-entrant.

The assembler is single pass so that each line of source code is translated into object code completely and on a line by line basis. Several cells within page Ø are used by the assembler to record its current state for a user and the object code produced by the assembler is always left in page 1. The top part of page 1 is used by the assembler to provide certain automatic programming facilities for the user and cell 10 protects this area by containing a MOCKP word which points to the last word which is available for normal use.

A source program line consists of the address of the location which is to receive the translated version of the source word followed by the source word itself. This source word can be any of the four types (but certain conventions must be kept to if the program is to list correctly.)

P-word

Example

Source: 100 TAKE 3,*260'(4); COMMENT

Object: 100 |0|10|001 000|011|100|1|1| 01000 00100|

The basic format for a P-word consists of a mnemonic terminated by a space followed by an accumulator address (an integer from 0 to 7) terminated by a comma followed by a store operand specifier. In its simplest form the store operand is a numeric address but this can be preceded by an asterisk if indirect addressing is required and terminated by an apostrophe if the user wishes to access at this point data from another page. If index modification is required, the store operand specifier is followed by the address of an index in brackets. A comment (ignored by the assembler) can be placed on the same line if it is separated from the rest of the source word by a semicolon.

Alternatives i) The accumulator address and terminating comma can be omitted if the user wishes to use accumulator 1.

ii) If the page bit is not set, the numeric store operand address can be replaced by an identifier leaving the assembler to allocate storage from the top end of store.

Example

Source: TAKE*PEAR(3)

Semi-translated version: TAKE*1022(3)

In this case the identifier itself has been stored in 1023 as an S-word and 1022 has been reserved as the workspace associated with this identifier.

Identifiers are normally introduced via a DECLARE command but can be declared by default (with a warning output to the user). Whenever an identifier is declared, space is reserved for it at the top end of the store and the content of cell 10 - the protection register is adjusted.

iii) An instruction can be written in such a way that the store operand itself is specified rather than its address.

Example 1

ADD 4,+3.6

3.6 is called a literal and the user here is instructing the assembler to organise the store so thar at run time 3.6 (an F-word) will be aided to the content of acc 4.

An F- or I-word literal is recognised by the initial + or - sign which must be present and an S-word by the initial quote. If a literal is given in octal, the initial open bracket is used to signify the presence of a literal.

The assembler copes with a literal by assigning a location from the top end of store, substituting its address into the instruction in place of the literal and placing the literal itself into the location assigned. Multiple calls for the same literal use the same copy from the top end of store

Example 2

Source	Semi-translated version
200 ADD+1	200 ADD 1023
201 SKAE 4,"x"	201 SKAE 4,1022
202 SUBT 3,(I00000001)	202 SUBT 3,1023
203 MULT 2,+1.0	203 MULT 2,1021
	1021 +1.0000
	1022 "*&&&"
	1023 +1

When the literal is another F-word it must be enclosed in square brackets

Example 3

Source	Semi-translated version
100 TAKE 2,[JUMP 16]	100 TAKE 2,1023
	1023 JUMP 16

P-word literals can themselves have literals thus:

Example 4

Source	Semi-translated version
100 TAKE 2,[TAKE+4]	100 TAKE 2,1023 1022 +4 1023 TAKE 1022

iv) Extracodes have their own special mnemonics so that the assembler regards EXTRA 3,1 and SQRT 3, as equivalent.

v) An instruction can be labelled by preceeding it by any identifier terminated with a colon. In this way forward jumps can easily be dealt with:

Example

Source	Semi-translated version
100 JUMP*LAB1 ------- 200 LAB1: TAKE 2,500(2)	100 JUMP*1022 ------- 200 TAKE 2,500(2) ------- 1022 NIL 200 1023 "LAB1"

Identifiers used with jump instructions do not need to be declared. Room is reserved for them in the usual way when they first appear either as a label or a reference to a label in a jump instruction.

vi) A user may call his own labelled subroutine by typing by itself the label after the usual location number thus:

100 CBRT

the assembler interprets this to mean the same as

100 JUMP 7,*CBRT

so that the link (MOCKP 101 in this case) is put on top of the stack and is followed by a jump to the address specified by the NIL instruction stored in CBRT as usual.

The assembler will not recognise the name of a label used in this way if it coincides with any of the mneumonics used for the functions or extracodes.

F- & I-words

These must be signed and are normally[1] not output when a program is listed. It is recommended that single constants are incorporated into the program as literals. If a table of constants is required these can be entered in a separate page or alternatively they can be placed manually at the top end of store with C(10) appropriately adjusted afterwards.

[1]PAGE parameters are always listed even if they are I-words.

S-words

These must appear in quotes. An S-word is normally only listed if it is a member of an S-word sequence following a CAPN extracode or one of the pair of S-words following a PAGE extracode.

The assembler prevents or discourages the user from doing a number of silly things. Only those functions with codes 000 000 to 011 111 are meaningful when literals are used and mistakes in this direction are detected by the assembler. Normally, the assembler expects instructions to be entered in sequence and outputs a warning if this is not the case. However, the command EDIT leaves the assembler in a state where it can receive instructions in any order. BREAK allows a single break in sequence and leaves the assembler in a state where instructions are expected in sequence. The assembler does not normally expect the user to assemble into locations outside the range 16 to C(10). This restriction does not apply to I- F- or S-words, to MOCKP or MOCKS pointer words, to NIL instructi-ns or to any word given in octal.

THE COMMAND LANGUAGE

A command can occur at the beginning of a line at assembly time or immediately following an altmode at any other time. An altmode causes any running program to be aborted with the message

INTERRUPTED L= -----
UNUSED DATA: ------

displayed, "L=" is followed by the current value of the SCR ie. the instruction which was just about to be obeyed. "UNUSED DATA:" is only output if appropriate and is followed by the next 6 characters of unused data. All unused data is lost.

In the following list of commands, the command itself is followed by a typical set of parameters (ie. information which the command must work with). Unless it is otherwise stated, the commands refer to page 1. Other pages can only be edited by saving them and then loading them temporarily as page 1.

The following commands are used to control the system:

BREAK Initiates a new segment of program. Allows a break in sequence. Addresses terminated by a "/" are relative to the first instruction after the last BREAK (If this first instruction is Ø/ this is an indication that no break in sequence has occurred so that the next address in sequence becomes the new base address.)

BREAK is also used to change back the condition of the assembler from EDIT mode.

EDIT After EDIT, the computer is ready to receive instructions without regard to sequence.

RUN Causes the current program to run from the first instruction typed in. (The address of this instruction is stored as a MOCKP word in cell 11). RUN 39 causes the program to run from the instruction in 39 regardless of the content of 11 and RUN 39:2 causes the program to run from the instruction 39 in page 2.

The hardware stack pointer is set so that the stack starts empty and the quotes marker is set to Ø.

CONTINUE Causes the current program to continue from where it left off, ie., the SCR is not reset. CONTINUE 39 causes the SCR to be reset so that the program continues from the instruction in 39 of page 1 with the stack pointer and quotes marker unchanged. CONTINUE 39:2 continues from location 39 in page 2.

TAPE Allows a program which has been prepared off line to be entered via the tape reader on the teleprinter.

KEY All tapes which are prepared off line should be terminated with the command KEY followed by a carriage return line feed.

ADVANCE +3,120,230. Moves forward the content of page 1 between 120 and 230 (inclusive) by 3 places. The operands of P-words pointing to words that have been moved are automatically adjusted. The first parameter (+3 in this case) must be a signed integer. If it is negative, part of the store is moved backwards.

NEW FRED. Page 1 is replaced by an empty page (ie. a page filled with I-word zeros) except for cell 8 which contains "FRED" the name of the page.

NEW FRED:3 replaces page 3 instead of page 1 but otherwise the command is the same. NEW FRED:Ø is not allowed.

NEW :3 is an instruction to the computer that page 3 is no longer required.

OLD FRED. Page 1 is replaced by a copy of a page called FRED already stored by the user on backing store. A user can obtain (if it is not protected) a copy of another user's filed page by typing OLD FRED [300,102] where 300,102 is the other user's project programmer number. Alternatively, a page from a public library of programs can be retrieved by the command OLD FRED*. Other pages can be loaded in this way so that the variants OLD FRED:2 and OLD FRED*:2 are legal.

RENAME FRED. This command renames page 1 with the name FRED. RENAME FRED:2 renames page 2 but RENAME cannot be used to rename pages kept on backing store or page ZERO.

SAVE FRED. This command saves on backing store a copy of the current page 1 if necessary renaming it first. SAVE FRED:2 saves a copy of the current page 2. If a file already exists called FRED, an error message is output. Page Ø cannot be SAVEd.

REPLACE FRED. This command is similar to SAVE but is only valid if a page called FRED is already on backing store. The existing page on backing store called FRED is replaced by a copy of the specified page in core.

UNSAVE FRED. This command cannot be used on current core pages. It is only used to remove a page on backing store called FRED.

LOGOUT The system responds with DO YOU MIND IF YOUR ERROR STATISTICS ARE RECORDED ON THE SYSTEM? When the user responds with YES or NO, control is returned to the PDP-10 monitor.

The following commands are used to provide diagnostic help:

TRACE)
CHECKA)
CHECKN) All three of these commands set the monitor bit to a one on selected instructions. If no parameters are given, the monitor bit is set on all jump instructions in page 1. If a list of addresses follow (preceded by a space and separated by commas) then the monitor bit is set on these locations. If the command is followed by a <u>single</u> identifier then the monitor bit is set on all instructions using this identifier.

When, in running a program, the computer attempts to obey an instruction with the monitor bit set, an interrupt is caused and control passes to the monitor The monitor obeys the instruction on behalf of the user but before doing so, diagnostic output to the user is given. The details of this output depend on whether TRACE, CHECKA or CHECKN was used last, the details are given below:

TRACE: The location of monitored instruction is output terminated by an asterisk.

CHECKA: A new line is output followed by an asterisk, the location of the monitored instruction and the content of the specified accumulator before the instruction has been obeyed.

CHECKN: A new line is output followed by two asterisks, the location of the monitored instruction, the content of the specified accumulator before the instruction has been obeyed, the address of the store operand and its content before the instruction has been obeyed. e.g.:

```
**L=163 C(3)="ABCD" C(1003)=[1]      +3
```

might be output if CHECKN TEMP had been used before an instruction such as

```
163 SKET 3,*TEMP
```

had been obeyed.

Monitor bits are normally only set on page 1.

[1] C(1003:2) is output to signify that the store operand is in location 1003 of page 2.

REMOVE This command removes the monitor bit set up by TRACE, CHECKA or CHECKN. REMOVE on a line by itself removes the monitor bit on all instructions and REMOVE 21, 39 removes all monitoring from instructions in the range 21 to 39.

LIST Causes an annotated copy of the current program in page 1 to be output on the teleprinter together with the necessary command words for the program to be successfully re-input. LIST 23 lists the single line 23 and LIST 23,49 lists the segment of program from 23 to 49. LIST 23,49:2 lists the appropriate locations from page 2.

DUMP Similar to LIST but the content of all locations within the stated range are output (apart from I-word zeros) regardless of whether or not they form part of the program.

The following commands provide high level programming facilities:

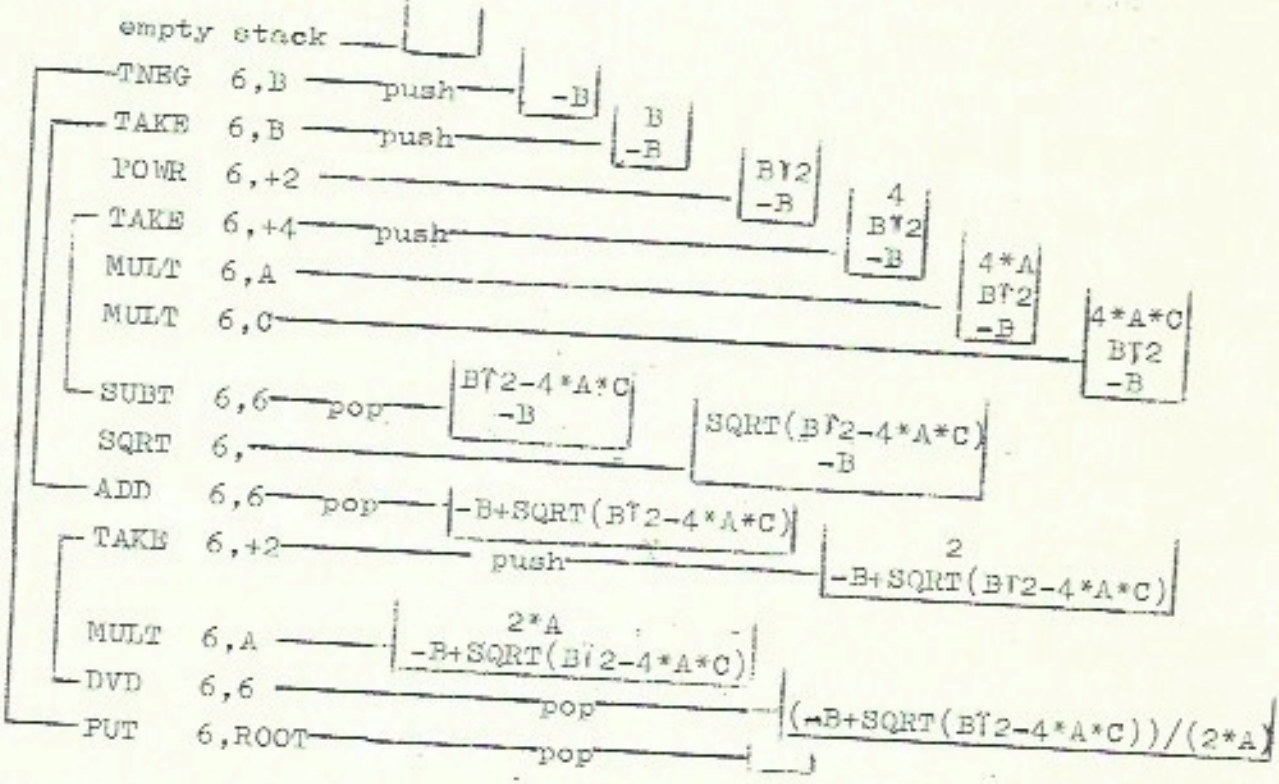
DECLARE Followed by a list of identifiers separated by commas, this command causes the computer to reserve locations at the top end of the page for variables with the given names. Identifiers associated with jump instructions and labels need not be declared but other identifiers used without declaration cause a warning to be output. The allocation of space for identifiers starts at cell 1023 and works downwards. Two adjacent cells are set aside for each identifier. One with an even address is used as work space for the identifier and the one immediately following is used to store the identifier's name.

PLACES+4 This command alters the places setting to 4 so that in future I- and F- words will be printed to 4 figures (if possible). When considering page layout, a user should remember that besides the 4 digits of the number the computer outputs 3 more characters so that space must be allowed for 7 characters altogether. The places setting cannot be greater than 8 or less than -8. Negative and zero settings have special effects which are described under the section on PRINT.

ASSIGN ROOT:=(-B+SQRT(B↑2-4*A*C))/(2*A)
This command sets up BBC code to assign the appropriate value to ROOT and this code is added to the end of the user's program. It is listed to complete the user's assembly version. Identifiers may not be used in an assignment statement before they have been declared. Single length multiplication (MULT) and division (DVD) are used and any of the extracodes or labelled

user subroutines can be used as functions although some extracodes are not particularly appropriate.

This particular example would cause the following code to be compiled with [ ] showing the content of the stack at each stage.



One dimensional array variables may be included within assignment statements (with the subscript in square brackets, but the responsibility of storage allocation for such array variables rests with the user. In particular, there is no check on subscript overflow.

Before the compilation of an arithmetic expression which contains subscripted variables, the user must make sure that the workspace of each array variable contains the appropriate (constant) pointer word.

<u>Example</u>

ASSIGN B[N]:=B[N+1]+B[B[N-1]+2]↑2

A user wishes to compute:

Let us suppose that the value of a subscript lies between 90 and -10. The user decides that he wishes to use cells in the current page with addresses from 500 to 600 for the array so that before the assignment statement is compiled he includes, within his program the statement pair:

```
TAKE [MOCKP 510]
PUT  B
```

B contains this pointer word throughout this section of the program and reference to array elements is made by

indirect addressing on B and with the value of the subscript providing index modification. The full code that would be compiled for the assignment statement is given below. The INT extracode which is included in each case before a subscripted variable is accessed, ensures that index modification with an F-word is not attempted. When reading through this code, the user is advised to consider the case when the content of N is 17.

```
empty stack ——[    ]
TAKE  6,N-push——[N ]  ie. 17
ADD    6,+1 ————[N+1]  ie.18
INT   6,
TAKE  6,*B(6)-pop & push—[B[N+1]]   ie. eighteen is
                                    replaced by
                                    C(510+18)
TAKE  6,N ———— push ————[N
                        B[N+1]]
SUBT  6,+1 ————————————[N-1
                        B[N+1]]
INT   6,
TAKE  6,*B(6)-pop & push—[B[N-1]
                         B[N+1]]
ADD   6,+2 ————————————[B[N-1] +2
                        B[N+1]]
INT   6,
TAKE  6,*B(6)-pop & push—[B[B[N-1] +2]
                         B[N+1]]
POWR  6,+2 ————————————[B[B[N-1] +2]↑2
                        B[N+1]]
ADD   6,6 —— pop——[B[N+1] +B[B[N-1] +2]↑2]
TAKE  6,N ———— push ————[N
                         B[N+1] +B[B[N-1] +2]↑2]
PUT   6,*B(6) ——pop & pop ——[    ]
```

4 IF This command provides high level conditional jumps. Any pair of arithmetic expressions can follow IF but they must be separated by a relational operator (one of =, ≠, <, >, ≤ or ≥) and terminated by either a colon followed by a label or a pair of colons followed by a third arithmetic expression. The following examples clarify its use:

<u>Example 1</u>

IF X+3 ≤ B[X] :L1
means jump to the label L1 if X+3 ≤ B[X]
and carry on with the program in sequence otherwise.

Compilation is as follows

```
TAKE  6,X ——push——  [ X ]
ADD   6+3 ————————  [ X+3 ]
TAKE  6,X ———— push ——  [ X ]
                        [ X+3 ]
TAKE  6,*B(6) ——— pop & push ——  [ B[X] ]
                                 [ X+3 ]
SUBT  6,6 ———— pop ————————  [ X+3-B[X] ]
PUT   6,5 ———— pop ——  [ ]
JLZ 5,*L1
JEZ 5,*L1
```

It is important to notice that the final test is made using accumulator 5, not the top of the stack. This is necessary because the state of the stack must be unchanged at the end of the statement but it does mean that any other value in accumulator 5 is lost.

<u>Example 2</u>

IF $X \leq 3$::200+X

means that if $X \leq 3$ jump to the instruction whose address is 200+X otherwise carry on in sequence. The code which would be added to the user's program in this case is now given assuming that his previous piece of program finished at 69.

```
70  TAKE  6,X — push —  [ X ]
71  SUBT  6,+3 ————————  [ X-3 ]
72  PUT   6,5 —— pop ——  [ ]
73  JGZ   5,78
74  TAKE  6,+200 —— push ——  [ 200 ]
75  ADD   6,X ————————  [ 200+X ]
76  INT   6,
77  JUMP    0(6) —— pop ——  [ ]
```

LOOP
REPEAT

This pair of commands enables a user to specify how he wishes sections of his program to be repeated a number of times. Instead of writing instructions at assembly level to cause his program to loop he can state the appropriate conditions using similar notation to that used with ASSIGN and IF. The basic format of the LOOP command is of the form:

LOOP A:=B:C:D

This compiles code to enable a peice of program to be repeated D times altogether, Each time through A takes different values, B the first time, B+C the second time, B+2*C the third time and so on.

A is called the counting variable, B is A's initial value, C is the step size and D is the number of times.

B, C and D can be constants, simple or array variables or arithmetic expressions of almost any complexity. A must be either a simple or an array variable.

After the user has typed in a LOOP command and has received in response the corresponding low level code added to his program, he will probably wish to add the instructions inside the loop. These can be added at assembly level or by using assignment statements. A user might even wish to nest a number of LOOP instructions inside one another, the only limitations are that the loops should be properly nested (ie. all inner loops must be terminated before an outer loop is), care should be exercised if a jump leads from inside a loop to outside and a jump should never lead from outside a loop to the inside.

When the instructions within the loop are complete a user closes it by typing the command REPEAT followed by the name of the counting variable. This causes a single jump to be compiled closing the loop. At this stage also the address of the exit instruction in the code that was compiled in response to the corresponding LOOP is completed because it is only at this stage that the end of the whole loop is known.

Occasionally a user may wish to change the step size or the number of times through the loop with each repeat. This he can do by inserting code within the loop to change the value of one or more of the variables which make up either the step size or the number of times. For this to be effective, it is necessary for the compiled instructions to evaluate afresh the step size and the number of times in case any such alterations have been made. More often such alterations are not made and the compiling routine takes advantage of the fact that it is impossible when simple constants are used.

<u>Example</u>

Suppose that in a previous section of program the user has set up values in B C and D. This section finished with an instruction in 59 (say) and now he types:

```
LOOP  A:=B+2:D↑4:3*C
```

The computer responds to this by typing back the following code:

```
60  TAKE  6,B  }
61  ADD   6,+2 }  Sets A equal to its initial value
62  PUT   6,A  }      of B+2
63  TAKE  6,0     Initiates a value on the stack for counting.
64  JUMP  69
65  TAKE  6,D  }
66  POWR  6,+4 }  Works out the step size
67  ADD   6,A  }  Adds the step size to A (but not the first
68  PUT   6,A  }      time round the loop).
69  TAKE  6,+3 }  Evaluates the number of times the loop
70  MULT  6,C  }      is to be repeated.
71  INT   6,      Make sure the number of times is an integer.
72  SKEI  6,6     Compares the top two items on the stack.
73  JUMP  76      A skip terminates, otherwise the count is incremented.
74  PUT   6,0     Finished. The stack must be left as it was found.
75  JUMP  0       Address not yet inserted.
```

Let us suppose that at this stage the user adds ten instructions of his own, placing the last one in location 85. To close the loop he now types:

REPEAT A

and the computer responds by typing back
86 JUMP 65
to close the loop and by completing the instruction in 75 so that it reads:
75 JUMP 87
The amended version of 75 is not retyped for the user.

If the step size had been the constant 7 (say) the instructions in 65 to 68 would have been replaced by:

```
TAKE  5,+7
XADD  5,A
```

If the number of times had been the constant 5 (say) then it is more convenient to count the number of times round the loop by counting down from 5 (5,4,3,2,1) instead of counting up (1,2,3,4,5) as hitherto. This means that the instruction in 63 is replaced by TAKE 6,+5 and 69-75 and replaced by

```
PUT  6,5    count removed to accumulator 5
JZD  5,0[1] exit if zero otherwise decrease count by 1.
TAKE 6,5    put count back on the stack.
```

[1] As before, the exit address is added later.

The reader should work through this example by reproducing the content of the stack at each point. He should notice that:

i) When the loop is completely finished, the content of the stack is the same as it was at the beginning of the loop.

ii) While the loop is in progress the count is kept on the stack from one time round to the next.

iii) This design is sufficient in itself to enable loops to be nested but a jump out of a loop before it naturally terminates means that the top item on the stack must

be manually removed with the instruction PUT 6,0 in cases where outer loops are to be allowed to continue.

6 MESSAGE "MARY HAD A LITTLE LAMB."

This command compiles the following caption on to the end of the user's program:

```
89  CAPN
90  "MARY"
91  " HAD"
92  " A L"
93  "ITTL"
94  "E LA"
95  "MB.&"
```

The message itself must not contain any quote signs or a new line.