Some coding conventions, subroutines, and alterations in the order structure.

This is a collection of miscellaneous information for the guidance of people writing codes for the machine, particularly in cases where the assembly program, AP1, is in use. The most important changes in order structure are stated in Section 1: these mostly have the nature of additions to the "Errata and Addenda to the Computer Manual" (March 1959) and have been obtained by slight modifications in logical design. Symbolic programs based on Memorandum #2 (July 1959) should be little affected; absolute codes of class 1 and 2 may require revision.

It is felt that the changes made are sufficiently useful to warrant modification of AP1 (January 1959) to accept a revised mnemonic list and operation structure. The proposed changes are symbolically helpful, and can easily be fitted into the existing AP1 codes without unduly extending the mnemonic table. The results are given in Section 2. We have also determined some further subroutine conventions, in line with Memorandum #2, which are aimed at relieving the problem of fast register usage.

Section 3 gives a list of subroutines in existence or near existence at the time of writing, and a revised list of AP1 mnemonics.

1. Changes in Machine Codes

Field 1: No change

Field 2; Class 0: Control Orders. The rule concerning the transfer of (CC) to PF is now: The contents of CC are transferred to PF if (and only if) OP2 = 4. Transfers to P2 (the second pathfinder) remain as before.

Field 2; Class 1: Arithmetic Orders. The basic codes (given by OP2) remain unchanged, except that exponent manipulations have been completely eliminated from the fixed point orders. Some options are now offered in OP1 and OP4 to complete the range of orders (all combinations of U and S are now possible) and define the rounding function. These options are taken by placing a '1' in the bit position indicated in the list below, which is given in decoding sequence:

(a) OP4 bit 1 : interchange (U) and (S)

(b) OP1 bit 1 : clear R_M to sign of U_M

(c) OP1 bit 2 : interchange (U) and (R)

(d) Execute arithmetic operation given in OP2

(e) OP1 bit 3 : round result in U

(f) OP4 bit 3 : store (U) in memory cell whose address is in I.

All floating point results are normalised. The rounding mode switch will still force rounding if it is 'on', even though OP1 bit 3 may be zero.

Field 2; Class 2: Store, Substitute, Set Tag. In the decoding sequence, OP1, OP2 and OP4 are defined as follows:

OP1 = 0 : no action

1 : $(S)_{1,39} \rightarrow U_{1,39}$

2 : $(S)_{1,6} \rightarrow U_{1,6}$

3 : $(S)_{1,27} \rightarrow U_{1,27}$

4 : $(S) \rightarrow U$

5 : $(S)_{40,54} \rightarrow U_{40,54}$

6 : $(S)_{7,54} \rightarrow U_{7,54}$

7 : $(S)_{28,54} \rightarrow U_{28,54}$

OP2 = 0 : Clear ATR (Arithmetic Tag Register : this has been set during execution of field 4)

1 : Set ATR = 1

2 : Set ATR = 2

3 : Set ATR = 3

4-7 : No action on ATR

OP4 = 0 : Store (U) and (ATR) in $L(I_{40,54})$

1 : Do not store

2-7 : (not to be used).

Field 2; Class 3: No change

Field 2; Class 4: The setting and clearing of special registers has been extended as follows:

OP1 = 2 : Set lights designated by 1's in $I_{40,54}$

6 : Store $I_{40,54}$ in some special purpose register.

If OP1 = 2, OP3 is decoded as:

OP3 = 0 : Sense lights on

1 : Mode lights on

2 : Trapping lights on

3 : Indicator lights on

4 : Sense lights off

5 : Mode lights off

6 : Trapping light off

7 : Indicator lights off

If OP1 = 6, OP3 is decoded as:

OP3 = 0 : Second pathfinder, P2 (77770)

1 : Sense light register, SL (77771)

2 : Increment register, X (77772)

3 : Mode light register, ML (77773)

4 : Trapping light register, TL (77774)

5 : Indicator light register, IL (77775)

6 : Output location register, B8 (77776)

7 : Input location register, B9 (77777)

Field 2; Class 5: As in the arithmetic orders, a "store" option is provided here after the logical operation.

Thus OP4 = 1 (Bit 21): Store (U) with (ATR) after operation.

Field 2; Class 6: No changes

Field 2; Class 7: There are no class 7 orders now. The 'NOP' is the class 2 : 20001.

Field 3: The useless code '10' (Store R in Z) has been re-interpreted to mean "Store Z in R", i.e., clear R to 54 zeros.

Field 4: No changes.

2. Changes in the Assembly Program

Many of the changes of section 1 do not affect the symbolic forms of orders. In some cases, a revised interpretation is appropriate.

OPN field; Class 0 (logical orders). By introducing a new code "TSR" (Transfer to Subroutine) a distinction is drawn between transfer orders which reset PF (TSR), and those which do not (TRA). Thus, on 'internal' unconditional transfers; TRA is now symonymous with SCC (set control counter), and it is possible to distinguish both conditional and unconditional transfers to subroutines (although the former are less general, since they may not include a test in OP2). e.g.

TSR SINH

and IF(TG2) TSR SINH

both reset (PF) to (CC) (upon making the transfer.)

Class 1. Arithmetic Orders. Names have been provided for some of the more useful options on division. In the notation of Programming Memorandum #2, Lecture 2, we now have:

Addition; ADD; $U_M' = U_M + S_M$

Subtraction; SUB; $U_M' = U_M - S_M$

Multiplication; MPY; $U_M' + \theta R_M' = U_M \times S_M$

FIXED Division; DIV; $U_M' + \theta R_M'/S_M = [U_M + \theta R_M]/S_M, |R_M'|<|S_M|$

POINT Integer Division; IDV; $U_M' + R_M'/S_M = [U_M]/S_M, |R_M'|<|S_M|$

Reverse Division; VID; $U_M' + \theta R_M'/U_M = [S_M]/U_M, |R_M'|<|U_M|$

Reverse Integer Division; VDI;

$$U_M' + R_M'/U_M = S_M/U_M, |R_M'|<|U_M|$$

FLOATING Addition; FAD; $U'_F = U_F \overset{r}{+} S_F, R'_E = U'_E$

POINT Subtraction; FSB; $U'_F = U_F \overset{r}{-} S_F, R'_E = U'_E$

Multiplication; FMP; $U'_F = U_F \overset{r}{\times} S_F, R'_E = U'_E$

Division; FDV; $U'_F = U_F \overset{r}{\div} S_F, R'_E = U'_E$

Reverse Division: VDF; $U'_F = S_F \overset{r}{\div} U_F, R'_E = U'_E$

All these results are rounded and normalized. For unrounded answers, use octal codes.

In order to use the store option, the appropriate mnemonic code is followed by '+1'. (In this way it is handled by the existing AP1 program without change). For example, the code required to replace the vector y_i in cells Y to Y+100 by ky; where (T4) = k would be:

```
        SB4   d100
T4      FMP+1 Y+B4,B4-1
B4      IF(POS)TRA  CC-2
```

Class 2. Store, Substitute, Set Tag. There are now 80 orders in this class, all potentially useful. For symbolic purposes, we divide them into two sub-classes, the first of which is regarded as replacing part of U in a memory cell, and the second as bringing the portion of a memory cell to U, without disturbing the value in storage.

Let V be the final address formed in Field IV. Then we have:

Store; STO; (V)' = (U), (S)' = V

Fetch and Store; FST; (V)' = (U), (S)' = (V)

Replace exponent; RPE; $(V)' = (U)' = (U)_{1,6}(V)_{7,54}, (S)' = (V)$

Replace Mantissa; RPM; $(V)' = (U)' = (V)_{1,6}(U)_{7,54}, (S)' = (V)$

Replace Left Half; RPL; $(V)' = (U)' = (U)_{1,27}(V)_{28,54}, (S)' = (V)$

Replace Right Half; RPR; $(V)' = (U)' = (V)_{1,27}(U)_{28,54}, (S)' = (V)$

Replace Address; RPA; $(V)' = (U)' = (V)_{1,39}(U)_{40,54}$, $(S)' = (V)$.

No operation; NOP; $(U)' = (U)$; $(S)' = (S)$; $(R)' = (R)$.

Clear and add; CLA; $(U)' = (S)$; $(S)' = (S)$; $(R)' = (R)$

Bring exponent to U; BEU; $(U)' = (V)_{1,6}(U)_{7,54}$; $(V)' = (S)' = (V)$

Bring Mantissa to U; BMU; $(U)' = (U)_{1,6}(V)_{7,54}$; $(V)' = (S)' = (V)$

Bring left half to U; BLU; $(U)' = (V)_{1,27}(U)_{28,54}$; $(V)' = (S)' = (V)$

Bring right half to U; BRU; $(U)' = (U)_{1,27}(V)_{28,54}$; $(V)' = (S)' = (V)$

Bring address to U; BAU; $(U)' = (U)_{1,39}(V)_{40,54}$; $(V)' = (S)' = (V)$

Note that NOP has been placed here instead of in class 7, and CLA has also been placed in this class, as a slight speed advantage is gained over the old Class 5 order. A memory access is normally required to set up S correctly for all these orders, with the exception of STO. Note also that all the 'store' orders and the 'replace' orders clear to zero both tag bits in memory. If another setting is required, the 'set tag' option in OP2 should be used.

Class 4: Set special purpose registers. These orders are not all so important as to deserve a permanent entry as an OPN mnemonic. Only STX (set increment register) has therefore been added to the list.

Class 5: Logical orders. As in the arithmetic class, the store option is used by writing '+1' after the order. e.g., the code to clear all but bits 49-54 of the words in location P to P+ N-1 would be written

```
        SB1   77
        SB2   N-1
B1      AND+1 P+B2, B2-1
B2      IF(POS)TRA  CC-2
```

A revised list of OPN codes in AP1 is attached to this memorandum.

Field 3. The mnemonic "R → Z" is unchanged, but is interpreted as indicated in Section 1.

Library Routines. As a guide to what is available to coders, a partial list of library routines is given in the next section. It is partial only in the sense that, since the routines have been written in several places, full details of space requirements and fast register usage are not always known. However, the potential user may have confidence in their availability as working codes, and lack of complete details need not be a deterrent to including references to a library subroutine in some other code.

The basic conventions of subroutine usage are given on pages 22-23 of Programming Memorandum #2. We remark immediately that item (vi) in that list should now read:

"(vi) Transfer to a subroutine is normally made with the TSR order, thus setting PF correctly. Other unconditional transfers use TRA, SCC or ACC."

In order to use a subroutine in an AP1 program, it must be identified by the same name throughout the code, and at some point the appropriate library tape must be added to the symbolic input tape. The library tapes are in octal, hexad, or a 'condensed binary' form, which need not concern the coder since they are correctly interpreted by the AP1 processors, headed by the identifying name. If the coder obtains his assembled program in absolute form on paper tape, it will contain all subroutines, and these need not be reloaded at later execution runs.

Some 'overlap' in use of fast working stores by the main and sub-program is often unavoidable, and it is then essential for the coder to preserve the stores affected in the main program before entering the subroutine, by placing them in a 'safe' memory position. They should be restored on return from the subroutine. There are some obvious ways of doing this, but a useful technique may be mentioned here, since it is used by subroutines in the library. In this it is assumed that B6 gives the first word address of a block of free working stores, and these are used by the main routine both as an extension of the 'fast' temporary stores and as a repository for F-series registers before entering a subroutine. Then provided the rule is followed that B6 always "points" to a block of free cells (being advanced and decremented as more or less space is required), it may be used as a working storage index at any level. In this way, the need for including private working stores within a subroutine is avoided. For example, to save (B4) and (T6) before entering routine F23, we would write:

```
B4    STO   B6 + 2, B6 + 1
T6    STO   B6 + 2, B6 + 1
      TSR   F23
      CLA   B6 - 1, U → T6
      CLA   B6 - 2, U → B4
      AB6   -2
```

In later assembly systems, the above code may be generated, or equivalent closed subroutine linkages written, by two special macro-orders:

```
SAVE   B4,T6
TSR    F23
UNSAVE B4,T6
```

and in planning complex systems of routines, it is convenient to employ an abbreviation such as this until final details of subroutine usage have been worked out.

It should be noticed that some of the data input routines will use B6 as an indication of where to put the numbers being read in; they will advance (B6) by an amount corresponding to the number of stores used, and it is therefore important for the coder to insure that (B6) is reduced to its initial value if a data input routine of this type is used within another subroutine. The alternative procedure, of always retaining enough storage space for initial and intermediate data within a program, may still be used.

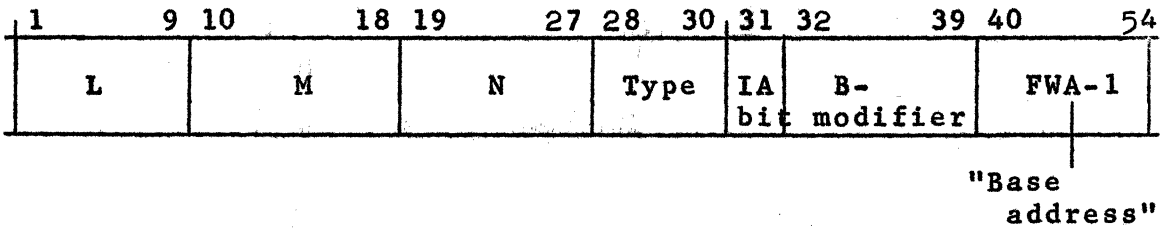
Provision is made for handling arrays of one or two dimensions "semi-automatically" by means of the type 1 codewords of Programming Memorandum #1. This leads to greater uniformity, shorter calling sequences, and little appreciable loss of time or space compared with conventional methods. For these reasons, only the "short forms" of subroutines are given in the following list. (The routines are also available with conventional calling sequences.) To summarize the codewords in their present form, it is assumed that each refers to a block of stores of length L, which may contain numbers of one sort or another, or more codewords. In the latter case, the IA (indirect address) bit is turned on in the codeword. If the consecutive stores contain elements $a_1, a_2, \ldots a_L$, then the codeword A contains a B1-modification bit, and has in its address part the location preceding the store containing a_1. Hence, it is always the case that to obtain a_i, it is sufficient to write the code:

```
SB1  i
CLA  *A.
```

Similarly, for matrices, to obtain the element $a_{i,j}$, it is sufficient to write:

```
SB1  i
SB2  j
CLA  *A.
```

The number L is given in bits 1-9 of A. Clearly, $1 \leq L \leq 511$. Bits 10-18 and 19-27 may give two parameters for use in indexing; these are denoted by M,N:



Finally, bit 28-30 may be used to indicate a <u>type</u> of array, which is used by various processing routines. Thus, we can interpret an array as a vector or matrix, or as a polynomial form of degree L-1:

$$P(A,x) = a_1 x^{L-1} + a_2 x^{L-2} + \ldots + a_L$$

or as a multi-length number:

$$M(A) = [\sum_{i=1}^{N} a_i^* \, 2^{-47i}] 2^{47M}$$

(where in the latter case a_i^* is the <u>mantissa</u> of a_i only), or as a member of any other set of objects between which useful mathematical operations may be defined.

As the system of codewords is refined, it should become unnecessary for the coder to know of their existence. They will form the basis of more automatic coding systems to be introduced. We give as an example a program P for reading two square (n x n) matrices A, B, and computing the matrix $A^{-1}B$, printing out the latter on the line printer:

```
P               ORG
        B6      RPA     P1
        PF      RPA     P2
                SB1     1
Z               TSR     READ, U→4
                TSR     MINV
        T4      STO     P3
                SB1     1
Z               TSR     READ, U→4
        T4      NOP     Z, U → T5
                CLA     P3,U → T4
        Z       TSR     MMPY, U → T6
        T6      SB1     1, U → T4
                TSR     PRINT
P1              CLA     a(B6), U → B6
P2              CLA     a(PF), U → CC
P3              OCT     0
                END
```

Here, B6 is used throughout the program, and the working storage list contains successively the matrices A, B, $A^{-1}B$ and their codewords.

It may be objected that by using codewords a firm grasp of the data in storage is lost. This is partly true, but it can readily be seen that by only two or three commands the "first word address" and "size" of an array can be obtained explicitly.

By the methods described above, subroutines may be incorporated in a program during assembly in a straightforward way. The output of the assembly process is (optionally) a condensed binary relativised self-loading tape containing the program. The assembly process is sufficiently fast to permit frequent re-runs during program checking, but as more complex assembly-compiling systems are produced, methods will be given for incorporating subroutines in the code immediately prior to execution.

As noted in Memorandum #2, arguments and results of any closed subroutines may be left either in the F-registers, or

the PF-list (i.e., the conventional calling sequences following the transfer order), or anywhere else addressed by one of these registers. We shall describe the subroutine using standard abbreviations for F-registers, and the notation P0,P1,...,Pk for members of the PF-list. Then let the notation:

$$(Y_1, Y_2,\ldots,Y_s)\ N\ (X_1, X_2,\ldots,X_t) \qquad (1)$$

denote the subroutine N with s output values and t input values, where the Y's and X's are generally chosen from the set of F-register names or the Pi's. Thus:

```
(T6) SIN (T6)
```

describes the single valued function SIN of the argument in T6, which leaves the value of the function in T6. It follows that to determine Y = SIN (F) we must code:

```
     CLA F, U → T6
     TSR SIN
T6   STO Y
```

Hence, the form (1) is adequate in describing the usage of N. It remains to give N a <u>meaning</u>, which is done in brief terms in the subroutine description, and to indicate which machine conditions, other than those described by (1), are liable to be changed by the action of N.

It is often convenient to name a collection of subroutines as a unit, and then provide access to each individual routine by means of a "subscript", which is really nothing more than a parameter stored in B1 before entering the routine. Such routines are denoted by:

$$(Y_1, Y_2,\ldots,Y_s)\ N_i\ (X_1, X_2,\ldots,X_t) \qquad (2)$$

Thus, in the case of

$$\mathrm{PRINT}_7(A,B,C,D)$$

the calling sequence would be:

```
SB1    7
TSR    PRINT
Z      A
Z      B
Z      C
Z      D
```

In the choice of N, we have tried to select the usual names for elementary functions. (These are mostly evaluated by means of economized polynominals derived at the Exploration and Production Research Division of the Shell Development Company.) Often, routines which effect the same transformation by different methods will have the same name. There seems to be no disadvantage in this as long as subroutine tapes are selected by hand, and it does have the advantage of keeping the set of different names to minimum size. Some subroutines, e.g. SIN and COS, are grouped together for obvious reasons, and appear on the same physical tape.

Not all possible functions have been placed in the library list, and many others are available. Some notes on these, and on the array routines which do not use codewords, may be seen at the Computer Project. However, it is often advantageous for the coder to write his own supplementary routines as they are required, in order to minimise overlapping in the use of fast stores.

3. Additions to the AP1 System

A summary of current AP1 mnemonics

In the following list, new or revised codes are underlined. Apart from recognising the familiar machine codes, the programmer may use the list in two other ways.

(i) For each operation mnemonic, a five digit octal code is given. The numerical digits define the essentials of the order code, but in addition to this the letter 'a' in an octal (or in some cases binary) digit position indicates that the corresponding digit is available for the synthesis of more complex orders in the same class. The conditions for effecting this synthesis are that in any two selected codes, numerical digits occurring in corresponding positions must be identical, and any other numerical digits must occur in positions in one code corresponding to "a" digit positions in the other. A digit containing "n" may not be used in synthesis with a numeric digit. By this rule, the codes 0 a a 5 a and 0 a a a 1 may be synthesised to give 0 a a 5 1. On the other hand 4 5 n a 1 and 4 5 n 5 5 may not be synthesised, since the last digits do not agree. Synthesis is made by taking the logical sum of all bits, treating 'a' and 'n' as zeros.

(ii) It is also possible for the machine to synthesise operation codes, although no economy of expression can be obtained when several order mnemonics are given. Thus, if by the rules above two or more octal codes may be correctly synthesised they may be written in the OPN field, separated by a comma:

e.g. ST0,ST1 ALPHA

IF(NMO,N≠E,NT3)TRA XMAX.

November 20, 1959

Revised Summary of API Mnemonics

Control

HTR	0 0 0 0 0	Unconditional halt and transfer
TRA	0 1 0 0 0	Unconditional transfer
TSR	0 1 4 0 0	Unconditional transfer to subroutine
SKP	0 2 0 0 0	Unconditional skip — Skip next instruction.
JMP	0 3 0 0 0	Unconditional jump by (X)
IF(test)HTR	0(a00)a a a	Conditional halt and transfer
IF(test)TRA	0(a01)a a a	Conditional transfer
IF(test)TSR	0(a01)4 a a	Conditional transfer to subroutine
IF(test)SKP	0(a10)a a a	Conditional skip (1)
IF(test)JMP	0(a11)a a a	Conditional jump by (X) (1)

If more than one test specified, action taken if any test satisfied - except if PNZ or NNZ used, then all tests must be satisfied.

(1) A floating point subtraction is carried out on all comparison orders.

Tests:

POS	0(0aa)1 1 a	Mantissa positive or zero
PNZ	0(1aa)1 5 a	Mantissa positive and non-zero
NEG	0(0aa)5 1 a	Mantissa negative or zero
NNZ	0(1aa)5 5 a	Mantissa negative and non-zero
MOV	0 a 2 a a	Mantissa overflow indicator on
NMO	0 a 6 a a	Mantissa overflow indicator off
EOV	0 a 3 a a	Exponent overflow indicator on
NEO	0 a 7 a a	Exponent overflow indicator off
ZER	0 a a 1 a	Mantissa zero
NZE	0 a a 5 a	Mantissa non-zero
EVN	0 a a 2 a	Bit 54 = 0
ODD	0 a a 6 a	Bit 54 = 1
SLN	0 a a 3 a	Sense lights on
SLF	0 a a 7 a	Sense lights off
NUL	0 a a 4 a	All 54 bits zero

Indicators are turned off.

Lights indicated by ADDR+MOD

TG1	0 a a a 1	Tag indicator 1 on	Specified indicators are turned off.
TG2	0 a a a 2	Tag indicator 2 on	
TG3	0 a a a 3	Tag indicator 3 on	
NT1	0 a a a 5	Tag indicator 1 off	Specified indicators are turned off.
NT2	0 a a a 6	Tag indicator 2 off	
NT3	0 a a a 7	Tag indicator 3 off	
NTG	0 a a a 4	No tag indicators on	

Arithmetic

ADD	1 n 0 n n	Fixed point add		Succeeding "+1" indicates use of store option.
SUB	1 n 1 n n	Fixed point subtract		
MPY	1 n 2 n n	Fixed point multiply	Double length No round	
DIV	1 n 3 n n	Fixed point divide		
VID	1 n 3 n 4	Reverse fixed point divide	Single length	
IDV	1 6 3 n n	Integer divide	Fixed point	
VDI	1 6 3 n 4	Reverse integer divide		
FAD	1 1 4 n n	Floating point add	Result in U rounded	
FSB	1 1 5 n n	Floating point subtract		
FMP	1 1 6 n n	Floating point multiply		
FDV	1 1 7 n n	Floating point divide		
VDF	1 5 7 n 4	Reverse floating point divide	Single length	

Fetch, Store

CLA	2 4 4 0 1	Clear and add to U	IM bit 1 = 0
BEU	2 2 n n 1	Bring exponent to U	
BMU	2 6 n n 1	Bring mantissa to U	
BLU	2 3 n n 1	Bring left half to U	
BRU	2 7 n n 1	Bring right half to U	
BAU	2 5 n n 1	Bring address to U	

NOP	2 0 0 0 1	No operation	
FST	2 0 4 n 0	Fetch and store	IM bit 1 = 0; tags unchanged.
STO	2 0 a n 0	Store	IM bit 1 = 1; tags cleared.
RPE	2 6 a n 0	Replace exponent in memory	IM bit 1 = 0; tags cleared unless combined with set tag operation.
RPM	2 2 a n 0	Replace mantissa in memory	
RPL	2 7 a n 0	Replace left half word in memory	
RPR	2 3 a n 0	Replace right half word in memory	
RPA	2 1 a n 0	Replace address in memory	
ST1	2 a 1 a a	Set tag 1	IM bit 1 = 0
ST2	2 a 2 a a	Set tag 2	
ST3	2 a 3 a a	Set tag 3	

Short registers

ACC	4 0 0 n n	Add to control counter
AB1	4 0 1 n n	Add to B1
⋮	⋮	⋮
APF	4 0 7 n n	Add to pathfinder
SCC	4 4 0 n n	Set control counter
SB1	4 4 1 n n	Set B1
⋮	⋮	⋮
SPF	4 4 7 n n	Set pathfinder
STX	4 6 n 2 n	Set X register

Shifts

UMR	4 1 n 1 a	U mantissa right
UML	4 1 n 2 a	U mantissa left
RMR	4 1 n a 1	R mantissa right
RML	4 1 n a 2	R mantissa left
DMR	4 1 n 1 5	Double mantissa right
DML	4 1 n 6 2	Double mantissa left

LUR	4 5 n 1 a	Logical U right	
LUL	4 5 n 2 a	Logical U left	
LRR	4 5 n a 1	Logical R right	
LRL	4 5 n a 2	Logical R left	
LRS	4 5 n 1 5	Long right shift	
LLS	4 5 n 6 2	Long left shift	
CRR	4 5 n 5 5	Circle right	$U_{54} \rightarrow R_1$ and $R_{54} \rightarrow U_1$
CRL	4 5 n 6 6	Circle left	$R_1 \rightarrow U_{54}$ and $U_1 \rightarrow R_{54}$
BCT	4 3 n 0 1	Bit count	

Lights, Modes

SLN	4 2 n 0 n	Sense lights on	Lights specified in ADDR+MOD
SLF	4 2 n 4 n	Sense lights off	
ERM	4 2 n 1 n	Enter repeat mode	ADDR+MOD field cannot be used.

Logical orders

AND	5 0 n n n	And to U	Preceeding "-" complements the result in U; succeeding "+1" indicates use of the store option.
ORU	5 1 n n n	Or to U	
SYM	5 2 n n n	Symmetric difference to U	
XTR	5 3 n n n	Extract S through R to U	

Input, output

PRk	6 2 n k n	Print with format k	k = 0,1,2,...,7
RHX	6 0 0 n n	Read hexads	
PHX	6 0 4 n n	Punch hexads	

Name	Function and Remarks	F-registers	Storage
(T4,T5)BINDC(T4)	Binary to decimal conversion. Fixed or floating point, depending on whether exponent bits are all 0 or not.*		
(T4)DCBIN(T4,T5)	Decimal to binary conversion, floating point form.*		
(PF,B1)TLU_1(P0,P1,T4)	A general logical table look-up, with argument T4, in the table whose codeword address is in P0, using the mask in P1. On exit, (PF)=0 if no equality was found; (PF)=1 and (B1) gives the relative address in the table if an equal entry is found.		
(T4)$MACOG_i$(B3,B4,B5)	Matrix codeword generator. i=type of matrix (1:storage by row; 2: by column; 3: upper Δ by row; 4:lower Δ by column; 5: lower Δ by row; 6: upper Δ by column); B3 = FWA; B4 = no. of rows; B5 = no. of columns.	All B's+B6list T4,T5. +B6 lost +TL	29

*See P.M. #2, p.30 for the appropriate decimal forms.

Name	Function and Remarks	F-registers	Storage
(T4,T5)CMPY(T4,T5,P0)	Complex multiply, floating point: T4' + iTS' =[T4* + iTS] x[(P0) +i(P0+1)]		
(T4,T5)CDIV(T4,T5,P0)	Complex divide, floating point: T4' + iTS' = [T4 + iTS]/[(P0) + i(P0+1)]		
(T4,T5)CVID(T4,T5,P0)	Reverse Complex divide, floating point: T4' + iTS' = [(P0) + i(P0+1)]/[T4 +iTS]		
(T4,T5)DPADD(T4,T5,P0) (T4,T5)DPSUB(T4,T5,P0) (T4,T5)DPMPY(T4,T5,P0) (T4,T5)DPDIV(T4,T5,P0) (T4,T5)DPVID(T4,T5,P0)	"Double precision" floating point routines. Each part of a number has the same exponent, so that (T4,T5) is interpreted as the number $T4+2^{-47}T5$. Similarly, P0 gives the address of the high order part of a double precision number stored in consecutive locations, i.e. $(P0) + (P0+1)2^{-47}$.		

Name*	Function and Remarks	F-registers	Storage
(T6)SIN(T6)	T6' = sin T6	T7,B5	73
(T6)COS(T6)	T6' = cos T6	T7,B5	
(T6)TAN(T6)	T6' = tan T6	T5,T7,B5	54
(T6)ATAN(T6)	T6' = arctan T6	T5,T7,B5	44
(T6)SQRT(T6)	T6' = $\sqrt{T6}$	A11T,B5	37
(T4,T6)EXP2(T6)	T6' = 2^{T6}	T7,B5	51
	T4' = 2^{-T6}		
(T4,T6)EXP(T6)	T6' = e^{T6}	T7,B5	
	T4' = e^{-T6}		
(T6)LOG2(T6)	T6' = $\log_2 T6$	T5,T7,B5	42
(T6)LOG(T6)	T6' = lnT6	T5,T7,B5	
(T6)E(T6)	T6' = E(T6)	A11T,B5	200 incl SQRT, LOG
(T6)K(T6)	T6' = K(T6)	A11T,B5	
(T6)J(T6)	T6' = J_0(T6)	A11T,B5	500 incl. SQRT, SIN, COS and LOG.
(T4,T6)JY(T6)	T6' = Y_o(T6)	A11T,B5	
	T4' = J_o(T6)	A11T,B5	
(T6)J1(T6)	T6' = J_1(T6)	A11T,B5	
(T4,T6)JY1(T6)	T6' = Y_1(T6)	A11T,B5	
	T4' = J_1(T6)	A11T,B5	

*Consult program files for a supplementary list of functions.

Name	Function and Remarks	F-registers	Storage
(B3)PRINT$_2$(T4,B3) (B3)PRINT$_3$(T4,B3) (B3)PRINT$_4$(T4,B3)	Dec output Oct output Hex output	B3 gives a type position before 0 after use. In AP1 form.	None
PRINT$_5$	Print 1 line, space, and interchange print matrices ready for next line.		None
PRINT$_6$(P0,P1,P2,P3,P4,P5)	Decimal output, 1 line, with 7 digits in fractional parts. The Pi contain addresses of data.		None
PRINT$_7$(P0,P1,P2,P3)	Decimal output, 1 line, with 13 digits in fractional parts. The Pi contain addresses of data.		None
(T4)READ$_2$ (T4)READ$_3$ (T4)READ$_4$	Dec input Oct input Hex input	In AP1 form. Each number is delimited by a comma, cr or tab.	None
(B6,T4)READ$_1$(B6)	General decimal input for a single array with codeword generation. The primary codeword is left in T4. Storage space is taken from (B6) onwards. Input matrices by rows. (1)		None
PRINT$_1$(T4)	General decimal output for a single array, whose codeword is in (T4).		None

(1) The B6 list is used only if T4 = 0 on entry. Otherwise, T4 is assumed to contain the codeword of some array region into which data is to be stored.

[N.B. In the function description, M(k) stands for the matrix whose codeword is k. Similarly, V(k) stands for the vector whose codeword is k. In the binary operations, if (T6) = 0, then space for the result is taken from the B6 list.]

Name	Function and Remarks	F-registers	Storage
(T4)MINV(T4)	$M'(T4) = [M(T4)]^{-1}$ By Gauss	A11B, A11T ML and TL	74 + (n-1) from B6.
(T6)MADD(T4,T5,T6)	$M'(T6) = M(T4) + M(T5)$		
(T6)MSUB(T4,T5,T6)	$M'(T6) = M(T4) - M(T5)$		
(T6)MMPY(T4,T5,T6)	$M'(T6) = M(T4) \times M(T5)$		
	$V'(T6) = M(T4) \times V(T5)$		
	$V'(T6) = V(T4) \times M(T5)$		
(T4)LINEQ(T4)	Solution of n-m sets of m simultaneous equations Ax=B where M(T4)=[A,B] stored by row. Solutions replace B portion of M(T4). By Gauss, with row exchange.		100+m from B6 list.
(T4)$JACOB_0$(T4,P0)	Reduction of M(T4) by modified Jacobi rotation method until modulus of largest off-diagonal element is <P0. M(T4) must be symmetric. Stored as upper Δ or rectangular, by row.		
(T4,T5)$JACOB_1$(T4,T5,P0)	As above, but with the formation of M(T5), a matrix of eigenvectors, simultaneously.		

[N.B. In the function description, P(k) stands for the polynomial whose codeword is k. In the binary operations, if (T6) = ℤ, then space sufficient for the resultant polynomial is taken from the B6 list.]

Name	Function and Remarks		F-registers	Storage
(T6)PADD(T4,T5,T6)	P'(T6)=P(T4)+P(T5)	(Floating point)	A11T,B123,X	133
(T6)PSUB(T4,T5,T6)	P'(T6)=P(T4)-P(T5)	" "	A11T,B123,X	plus
(T6)PMPY(T4,T5,T6)	P'(T6)=P(T4)xP(T5)	" "	A11B,T456,X	B6 list
(T6)PDIV(T4,T5,T6)	$\frac{P'(T6)}{P(T5)} + \frac{P'(T4)}{P(T5)} = \frac{P(T4)}{P(T5)}$	" "	A11B,A11T,X	
(T6)PVAL(R4,T5)	Evaluate P(T4) for the argument T5, by floating point arithmetic, leaving result in T6.	" "	B12 T456	8
(T5)PBAIR(T4,P0)	Determine real and conjugate complex roots to accuracy P0 by Bairstow method.	1	AUB,AUT, X,SL	147 plus B6 list.
(T5)PMULL(T4,P0)	Determine real and complex roots of complex polynomial P(T4) by Muller method. (1)			
(T5)PCHBY$_1$(T4,T5, P0)	Reduction of real polynomial P(T4) by Chebyshev procedure with | max error | < |P0| (2)			
(T5)PRGEN(T4,T5)	Generate real polynomial P(T5) with given real roots V(T4).			

(1) See J.H. Wilkinson, "The Evaluation of the Zeros of Ill-conditioned Polynomials", Num. Math. v1, part 3, (1959), p.150.

(2) When i=0, P(T4) is considered in the range (0,|T5|); when i=1, in (-|T5|,+|T5|).

[N.B. In the function description, N(k) stands for the multiple precision number whose codeword is k. All results are normalised so that the fractional part of the number is in the range (-1,1). Where there is insufficient space for the result of an operation, it is truncated at the low order end.]

Name	Function and Remarks	F-registers	Storage
(T6)MPADD(T4,T5,T6)	$N'(T6) = N(T4)+N(T5)$	A11B,A11T,X	351
(T6)MPSUB(T4,T5,T6)	$N'(T6) = N(T4)-N(T5)$		
(T6)MPMPY(T4,T5,T6)	$N'(T6) = N(T4) \times N(T5)$		
(T4,T6)MPDIV(T4,T5,T6)	$N'(T6)+\frac{N'(T4)}{N(T5)} = \frac{N(T4)}{N(T5)}$ where $\lvert N'(T4)\rvert < \lvert N(T5)\rvert$.		
(T4)MPSGL(T4)	Multiple precision to single precision floating point conversion. Result in T4.	T456,B1234	25
(T5)SGLMP(T4,T5)	Single precision (T4) to multiple precision form N'(T5).	T45,B1234	19

Corrections and Additions to P.M.#3

p.6 line 9 read: "to replace the vector y_i in cells Y to Y+100 by Ky_i where"

line 12 read: " T4 FMP+1 Y+B4,B4-1"

p.11 line 1 for "$a_{i,j}$" read "$a_{i,j}$"

p.12 line 5 read: " Z TSR READ,U→T4 "

line 9 read: " Z TSR READ,U→T4 "

p.17 Six lines up, delete: "IM bit 1 = 0"

p.20 Under 'F-registers' read: "All B's,T4,T5,+B6 list +TL"

p.21 For "TS" read "T5" throughout.

p.24 Four lines up read: "(T4,T5)$JACOB_1$(T4,T5,P0)"

p.25 For "(T6)PVAL(R4,T5)" read "(T6)PVAL(T4,T5)"

p.22 Elementary functions: Error and range. The functions listed here are evaluated for all values of the argument for which the (real) function is defined by first reducing the argument to a range in which an approximating polynomial expansion is known. The total error in the function therefore depends on the accuracy with which this initial transformation is performed. We list below the maximum absolute error, E, in function values in the range in which they are expanded:

sinT6 : $E = .2\times10^{-14}$, $-\frac{\pi}{4} \le T6 \le \frac{\pi}{4}$

cosT6 : $E = .3\times10^{-13}$, $-\frac{\pi}{4} \le T6 \le \frac{\pi}{4}$

tanT6 : $E = .25\times10^{-12}$, $-\frac{\pi}{2} < T6 < \frac{\pi}{2}$

arctanT6 : $E = .24\times10^{-12}$, $0 \le T6 < \infty$

$\sqrt{T6}$: $E = .8\times10^{-12}$, $1/2 \le T6 < 1$ (Newton)

2^{-T6} : $E = .55\times10^{-12}$, $0 \le T6 < 1$

$\log_2 T6$	:	$E = 1.1\times10^{-12}$,	$.5 \leq T6 \leq 1$
$E(T6)=E(k^2)$	:	$E = .5\times10^{-11}$,	$0 \leq T6 \leq .992$
		$E = .4\times10^{-12}$,	$.992 \leq T6 < 1$
$K(T6)=K(k^2)$	:	$E = .3\times10^{-11}$,	$0 \leq T6$ .9375
$J_0(T6)$	:	$E = .16\times10^{-12}$,	$0 \leq T6 \leq 4$
$Y_0(T6)$	:	$E = .24\times10^{-12}$,	$0 \leq T6 \leq 4$
$J_1(T6)$	:	$E = .02\times10^{-12}$,,	$0 \leq T6 \leq 4$
$Y_1(T6)$	:	$E = .5\times10^{-12}$,	$0 \leq T6 \leq 4$

For $4 \leq T6 < \infty$, the Bessel functions are evaluated by formulae of the type:

$$J_0(T6) = (T6)^{-1/2} \left[P_0(T6)\cos\left[T6 - \tfrac{\pi}{4}\right] - Q_0(T6)\sin\left[T6 - \tfrac{\pi}{4}\right] \right]$$

Where the E-term for the P's and Q's has maximum value 1.5×10^{-12}.