Rice Institute Computer Project

Programming Memorandum #2

July 1, 1959

Preliminary Notes on Programming

This is an attempt to summarise in as short a space as possible the basic features of the computer and should be used as a complement to the note on API, the assembly program. The Computer Manual remains the basic reference work, but it is hoped that most pieces of coding can be successfully tackled with an understanding of the following notes, and that a full study of the Manual can be left to a later stage.(1)

Our experience has shown already that the machine codes, although flexible enough to meet the demands of the most ingenious programmer, bewilder the newcomer by sheer proliferation, and it is correspondingly difficult to gain from the Manual an idea of the relative importance of various commands. Thus the mnemonics of API put a subset of the machine orders at the programmer's disposal, and this Note only deals with such orders. Similarly, the Trapping feature of the machine is not discussed here, and the use of tags is confined to the arithmetic tag indicators. Input and output is dealt with by subroutine where possible, and some subroutine conventions are outlined. No magnetic tape orders are discussed.

(1) By which time the Manual itself may have caught up with the most recent changes in the machine. In points where the three works are in conflict, the historical sequence of writing gives a clue to the probable accuracy of each account:

September 1958:	Computer Manual
January 1959:	API - A Basic Assembly Program
March 1959:	Errata and Addenda to the Manual

Lecture 1. Order and Number structure.

A definition of terms is appropriate. The machine is binary so that conversion between the base of 10 and 2 has to be programmed. It has electrostatic storage with an approximate access time of 8 - 10 microseconds independent of the storage location except for seven special "fast" stores with access time of 1 microsecond. There are 8191 words in storage, each of 56 bits.(1) Two of these bits are used as "tags", so it is correct to regard both instructions and numbers as 54 bit words with a "label" of 0, 1, 2 or 3.(2) Unless stated otherwise, the label is understood to be 0. The words in storage are numbered serially from 0000 to 8191 (decimal) or 00000 to 17777 (octal), and a word is referred to by its location number or address. Given M, an address in storage, the contents of M are denoted in what follows by (M). Given X, a word in storage, the location of X is L(X).

Thus, $$L((M)) = M.$$

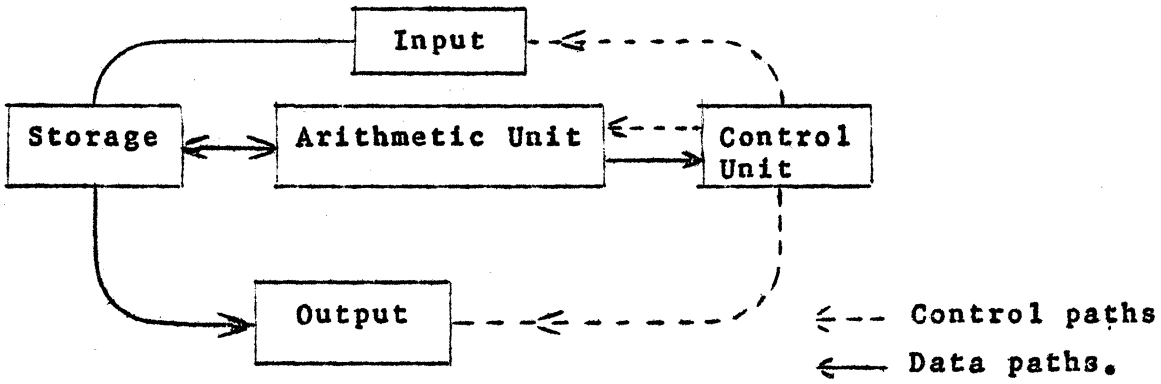
The machine operates in the serial mode, so that orders to be obeyed successively follow one another in storage, unless the first is a transfer order. However, the machine is parallel in the sense that all 54 bits are transferred from one register to another in a single step, along 54 separate lines.

Logically, an overall scheme of the machine divides into five sections:

---

(1) i.e., A digit taking only the values 0 and 1.

(2) i.e., Corresponding to the combinations 00, 01, 10, 11 for bits 55 and 56 (Tags 2 and 1 respectively).



The Arithmetic Unit has eight registers:[1] U(universal), R(remainder), S(storage), Z(zero, a "virtual" register of 54 0's) and four "fast" registers, T4, T5, T6, T7. Each has a regular machine address, i.e., 00001, 00002, 00003, 00000, 00004, 00005, 00006, and 00007 in the order given above. In addition to their special purposes they can act as "normal" storage locations. Their function is discussed in lecture 3. A distributor acts as an "exchange point" between all eight registers, and between storage and control unit. ARITHMETIC UNIT

We consider the (k + 1)-bit register whose digits are given by $m_s m_1 m_2 \ldots m_k$. Then if $m_s=0$, the number which this represents is determined by: BINARY ARITHMETIC

$$M = 2^q (m_1 2^{-1} + m_2 2^{-2} + \ldots + m_k 2^{-k}) \qquad (1)$$

where 2^q is some scale factor. Thus, if $q \leq 0$, M is a fractional number, and if $q \geq k$, M is an integer. On "fixed point" orders, the coder can control q so that any accuracy in M can be attained, although the machine orders proceed on the assumption that $q = 0$. If $q \neq 0$, it can be

(1) The word "register" is generally used to denote a storage position with some specialized purpose.

adjusted by the arithmetic shift orders. The form of (1) is sufficient to determine the shifts necessary for correct adjustment of q before any operation. On "floating point" orders the machine controls and adjusts q automatically. The first six bits of each numerical word are used to give a number E (the exponent) from which q can be found. The remaining 48 bits give M, the mantissa.

If $m_s = 1$, the number is negative. Its magnitude is found by first complementing each bit and then evaluating the new number as in (1); e.g., let $q = k = 4$:

	m_s	m_1	m_2	m_3	m_4	
then	0	0	1	1	0	represents +6
	1	1	0	0	1	represents -(00110) = -6
	1	1	1	1	0	represents -(00001) = -1
	1	1	1	1	1	represents -(00000) = 0.

There are two zero's in the system (given by all 1's or all 0's). As a consequence of this, addition must be performed with "end-around" carry:

	-6:	1	1	0	0	1	-5:	1	1	0	1	0
add	-1:	1	1	1	1	0	+7:	0	0	1	1	1
[carry →	1]	1	0	1	1	1	1]	0	0	0	0	1
						1						1
Answer	-7:	1	1	0	0	0	+2:	0	0	0	1	0 .

ONE'S COMPLEMENT NOTATION

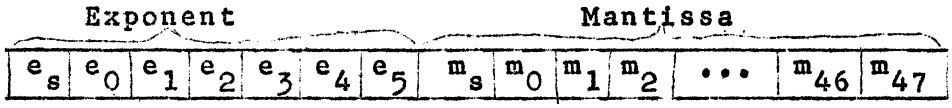
A formal definition, corresponding to (1), but including negative numbers, can be seen to be:

$$M = 2^q \left(-m_s + m_1 2^{-1} + m_2 2^{-2} + \ldots + (m_k + m_s) 2^{-k}\right) \qquad (2)$$

The form (2) requires $|M|<2^q$. Since in an addition operation this limit may be exceeded, an extra bit is provided to detect when "overflow" has occurred. This is placed between the sign and bit 1.

	m_s	m_0	m_1	m_2	m_3	m_4
+10	0	0	1	0	1	0
+ 7	0	0	0	1	1	1
+17	0	1	0	0	0	1

Overflow is sensed when $m_o \neq m_s$. It can be corrected by a right shift and an adjustment to q. Since U is the "accumulator", it is provided with two overflow bits, e_o and m_o for exponent and mantissa respectively.(1) The diagram shows the conventional labelling of bits in U. Other registers are similar, without e_o and m_o:



By convention, the mantissa M is evaluated by (2), with q=0, k=47. The exponent E is evaluated with q=k=5. The value of a floating point number is $M_o 2^{8E}$.

CONTROL UNIT

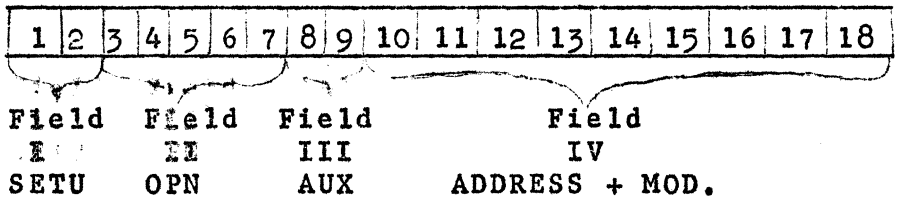
Orders are sequenced by taking successive words from storage to the instruction register, I, in the Control Unit. Each order is a 54 bit word, i.e., a word of 18 3-bit triads. The order is decoded in four fields, described below.

Two series of "fast" registers are employed: the arithmetic or A series mentioned above, and the B series

(1) In transfers to U, e_0 and m_0 are set equal to e_s and m_s, respectively. In transfers from U, e_0 and m_0 are ignored.

of 15 bit registers, which form part of the control unit. There are eight of the latter, referred to by the symbols CC (control counter), B1, B2, B3, B4, B5, B6, and PF (pathfinder). B1 through B6 are conventional index registers (Lecture 3). PF and CC are also index registers, but have additional functions (Lecture 3).

The diagram shows the sub-fields of I in terms of its 18 triads:



The order of decoding is:

(A) Field I (SETU) brings an A or B series register to U.

(B) Field IV (ADDRESS + MOD) brings a number to S.

(C) Field II (OPN) causes an operation to be performed on one or more of U, R, and S, generally leaving the result in U or U and R or storing the result in memory.

(D) Field III (AUX) results in some auxiliary operation being performed.

On completion of the order, (CC) gives the address of the next order to be obeyed.

Lecture 2. Basic Operation Codes.

From the Note on AP1, it is seen that an order is written symbolically in four fields on the coding form, corresponding to the sequence (ACBD) given at the end of Lecture 1. A fifth field, LOCN, gives the address of the order itself for cross referencing purposes. Any field, except OPN (Field II), may be blank. We summarise OPN first, assuming SETU has placed a number in U and ADDR + MOD has brought a number to S. The first triad of Field II gives the class to which the order belongs.

For each order we give its name; its mnemonic abbreviation; and its function. Let X_M denote the mantissa value of (X), X_E the exponent value (see Lecture 1) and X_F the floating point value = $X_M.2^{\theta X_E}$. Let a prime denote the value after the operation has been performed. Let θ stand for the constant factor 2^{-47}. When a word is regarded as a pattern of bits, rather than as a number, the bits are labelled 1,2,3...54 from left to right. Then any sub-field is indicated by a subscript notation, e.g., $(X)_{10-13}$ means bits 10,11,12,13 in location X.

Class 0: CONTROL

See Lecture 3.

Class 1 ARITH-METIC ORDERS

In all arithmetic orders U and S are involved. R is also used in multiplication, to hold the low order part of the product, and in division, to hold the low order part of the dividend before the operation, and the remainder afterwards.

Floating point addition and subtraction may shift part of the

smaller number into R. Floating point results are normalised so that

$$1 > | U_M | \geq 256^{-1}$$

Rounding on multiplication is controlled by a switch on the console. Normally, no rounding takes place. After floating point orders, if $U_E \leq -32$, exponent underflow has occurred and U is cleared to zero. If exponent overflow occurs, an indicator is turned on, but further action is up to the coder. Divide checks, caused when $|S| < |U|$ on fixed point orders, or possibly when a non-normalised floating point divisor is used, cause a machine halt, or may be ignored, depending on a console switch.

There are two groups of orders:

FIXED POINT

Addition; ADD; $U'_M=U_M+S_M$, $U'_E=S_E$

Subtraction; SUB; $U'_M=U_M-S_M$, $U'_E=S_E$

Multiplication; MPY; $U'_M+\Theta R'_M=U_M \times S_M$, $R'_E=S_E$

Division; DIV; $U'_M+\Theta R'_M/S_M=[U_M+\Theta R_M]/S_M$, $|R'_M|<|S_M|$, $R'_E=S_E$

FLOATING POINT

Addition; FAD; $U'_F+\Theta R'_F=U_F+S_F$, $U'_E=R'_E$

Subtraction; FSB; $U'_F+\Theta R'_F=U_F-S_F$, $U'_E=R'_E$

Multiplication; FMP; $U'_F+\Theta R'_F=U_F \times S_F$, $U'_E=R'_E$

Division; FDV; $U'_F+\Theta R'_F/S_F=[U_F+\Theta R_F]/S_F$, $|R'_F|<S_F$, $U'_E=R'_E$

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

Class 2 STORE ORDERS

Store; STO; $(V)'=(U)$

Store sum; STS; $(V)'=U_F+S_F-\Theta R'_F$

Replace left half; RPL; $(V)'=(U)_{1-27}(V)_{28-54}$

Replace right half; RPR; $(V)'=(V)_{1-27}(U)_{28-54}$

Relace "M" digits; RPM; $(V)'=(V)_{1-39}(U)_{40-54}$

Class 3 There are no orders in Class 3.

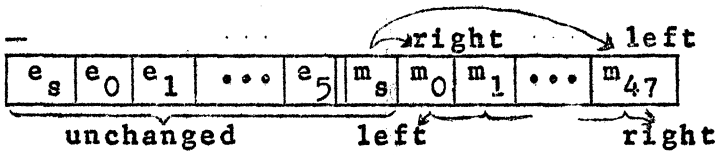
Class 4 SHIFTS

All these orders depend on the number, V, formed when Field IV is decoded. Only U and R can be shifted. Logical

shifts treat all 54 bits. Arithmetic shifts treat mantissas only, and have the property that a shift left of 1 place multiplies a number by 2, and a shift right of 1 divides it by 2, provided, on double length shifts, that the sign bits of $(U)_M$ and $(R)_M$ are equal.(1)

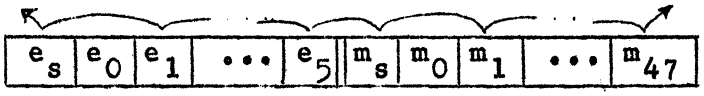
We can shift U, R, or both U and R (though not connected) as follows:

Arithmetic: U:



The sign bit determines whether 0 or 1 is brought into the vacated position. Except for the omission of the overflow bit, R is identical with U, for shifting.

Logical:

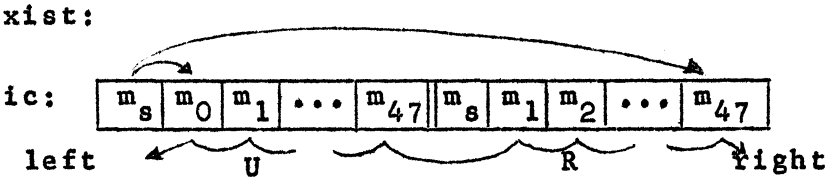


: U

The overflow bits are ignored; everything else shifts right or left. Empty spaces are filled in with zeros. The end bits are lost.

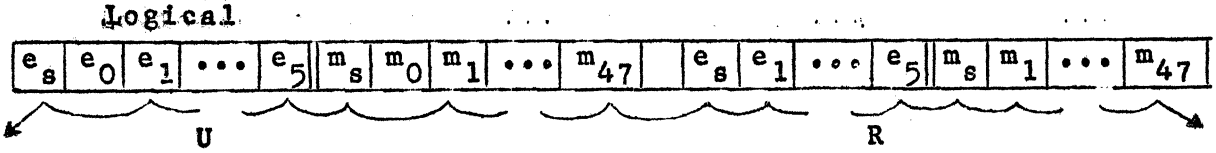
If U and R are defined to be connected, the following shifts exist:

Arithmetic:



The exponent bits are ignored; m_s of R is skipped; m_s of U fills in. U_{m_s} determines whether 0 or 1 will be brought to U_{m_s} (on right shifts) or R_{m_s} (on left shifts).

Logical



The overflow bits are ignored; spaces are filled by zeros.

(1) And provided significant digits are not shifted out of the register.

Class 4 SHIFTS

In a class 4 order, the number V gives the number of places to be shifted, and is evaluated modulo 128. Symbolically: Arithmetic: UMR, U mantissa right; UML, U mantissa left; RMR, R mantissa right; RML, R mantissa left, DMR, double mantissa right; DML, double mantissa left. Logical: LUR, logical U right; LUL, logical U left; LRR, logical R right; LRL, logical R left; LRS, long right shift; LLS, long left shift.

We also have the bit count: BCT; a logical shift of R right V places is made, and the number of ones shifted off the end is added into U.

Class 4 B-REGISTERS

In class 4 we can modify the contents of any B series register, using the 15 bit number V.

Set control counter; SCC; $(CC)'=V$

Set B register #1; SB1; $(B1)'=V$

etc.

Add to control counter; ACC $(CC)'=[(CC)+V]_{\text{mod } 2^{15}}$.

Add to B register #1; AB1; $(B1)'=[(B1)+V]_{\text{mod } 2^{15}}$.

Class 5 LOGICAL ORDERS

The basic three function tables are:

AND

(U) \ (S)	0	1
0	0	0
1	0	1

ORU

(U) \ (S)	0	1
0	0	1
1	1	1

SYM

(U) \ (S)	0	1
0	0	1
1	1	0

The complement tables are obtained if the OPN code is preceded by a "-" sign. Extract (XTR) is given by

(S)	(R)	(U)	(U)'	
0	0	0	0	
0	0	1	1	
0	1	0	0	i.e., R "masks"
0	1	1	0	S into U.
1	0	0	0	
1	0	1	1	
1	1	0	1	
1	1	1	1	

These orders are best understood by their commonest uses. AND uses a "mask" in S to clear portions of U to zero. ORU places the contents of S into the corresponding portions of U. SYM adds two numbers bit by bit without carry and can be used for the exact comparison of two numbers. XTR places a field from S, determined by a mask in R, into U without disturbing the rest of U. Note the particular case of ORU, namely Clear and Add; CLA; (U)'=(S).

Class 6
INPUT-
OUTPUT

See Lecture 5.

Class 7
SPECIAL

One special order is provided, No Operation; NOP; Field II has no effect. The other fields may be effective, however, and the full "no operation" must be written:

U NOP S .

Examples.[1] Let X1, X2, X3 be AP1 symbolic locations containing floating point numbers x_1, x_2 and x_3

(i) To form $x_1+x_2+x_3$ and store in X1:

```
CLA    X1      x1 -> U
FAD    X2      x1+x2 -> U
FAD    X3      x1+x2+x3 -> U
STO    X1      U -> X1
```

(ii) To form $x_1^2 + x_2^2$ and store in T4.

```
CLA    X1      x1 -> U
FMP    U       x1^2 -> U
STO    T4      U -> T4
CLA    X2
FMP    U
FAD    T4
STO    T4      x1^2+x2^2 -> T4
```

(iii) To store the exponent of x_1 in the exponent position of x_2.

```
CLA    X1
LUR    d48
LUL    d48
STO    T4      (X1)E -> T4
CLA    X2
ADD    T4      Fixed point add.
STO    X2      (X1)E(X2)M -> X2.
```

(1) In the examples given here, the first two columns give actual code. The third column of explanatory remarks is ignored by the machine.

Lecture 3. Control Orders and Fields I, III, and IV.

API allows Field IV to be determined by a symbolic or an absolute (decimal or octal) address. If symbolic, it is translated into an aboslute code by the assembly program. In general, the contents of this address are brought to S by the decoding of Field IV, and it then constitutes one of the operands in Field II. Thus,

CLA	X2	brings (X2) to S, then sends (S) to U
FAD	X3	brings (X3) to S, then floating adds (S) to (U) and leaves the answer in U.
LLS	5	brings (00005) to S, then shifts (UR) left logically 5 places.

Thus there is some redundancy in Field IV, and it is shown later how to eliminate this.

Class 0 CONTROL

The largest set of orders is Class 0, Control. API provides a small subset of these, in mnemonic form. They are of two types:

Unconditional Transfers which cause a break in the normal serial sequencing of commands:

Transfer; TRA; Take the next order from the address given in field IV

Halt and transfer; HTR; Stop, then proceed as in TRA when the "start" key is pressed

Skip; SKP; the next order is omitted.

Jump; JMP; the next (X) orders are omitted, where X is the 15-bit special register 77772_8.

Conditional Transfers, which cause a break in the sequencing of orders if some condition is satisfied; otherwise normal sequencing occurs. The general form is IF(XXX)TTT where TTT is one of TRA, HTR, SKP, and JMP; and XXX is the mnemonic for a condition given below. Generally this depends on (U), but it may refer to the sense light register 77771_8, or the indicator register 77775_8. The SKP and JMP transfers are dependent on $[U_F-S_F]$, which is placed in U before the test is made (note that (R) is lost in this case). S is always cleared to zero. The tests on (U) are then:

ZER; (U)=0? POS; (U)$\geq$ 0? NEG;(U)$\leq$ 0?
NZE; (U)$\neq$ 0? PNZ; (U)$>$ 0? NNZ;(U)$<$ 0?
EVN; Is the last bit of (U) = 0?
ODD; Is the last bit of (U) = 1?
NUL; Are all 54 bits of (U) = 0?

The tests on sense lights specify a 15-bit pattern in Field IV; they do not affect the status of the lights:

SLN; Are the sense lights denoted by 1's in Field IV all ON?
SLF; Are the sense lights denoted by 1's in Field IV all OFF?

The tests on the indicator register are as follows; they also turn off the corresponding indicator:

MOV; Has mantissa overflow occurred?
NMO; Has no mantissa overflow occurred?
EOV; Has exponent overflow occurred?
NEO; Has no exponent overflow occurred?

TG1; Is tag indicator no. 1 ON?
TG2; Is tag indicator no. 2 ON?
TG3; Is tag indicator no. 3 ON?
NTG; Are all tag indicators OFF?

Example (i) IF(NEG)SKP X1 :"If $[U_F - X1_F] \leq 0$ omit the following order. Otherwise perform the next in sequence".

(ii) IF(NMO)TRA THETA : "If no mantissa overflow has occurred transfer to the order in location THETA, Otherwise, proceed serially."

Field IV B-MODIFICATION

Of the nine triads in Field IV, 5 specify a 15-bit address. Before using the address, it may be modified by the (modulo 2^{15}) addition of one or more of the B series registers. Eight bits in Field IV are used to specify which B registers to use. Symbolically, the notation illustrated by "X5+B1+B6+CC" is used. An example shows the power of B-modification.

Example (iii) Find $y = \sum_{i=1}^{300} x_i$ where x_i are stored in consecutive locations starting in XF. One way to do this is:

```
START   CLA   XF
        FAD   XF+1
        FAD   XF+2
        -----
        FAD   XF+d299
        STO   Y
```

This takes 301 orders. A better way, taking 11 orders, is:

```
START       CLA   Z
            STO   T5
            SB1   d299
SUM         CLA   T5
            FAD   XF+B1
            STO   T5
            AB1   77776
            CLA   B1
          IF(NNZ)TRA  SUM
            CLA   T5
            STO   Y.
```

Any number of B registers may be used. They can be written down in any order, connected by "+" signs.

The description of the machine given up to this point is sufficient for many tasks and for practice the coder should try a few examples with the facilities at his disposal. It will be seen that with this command structure only marginal advantages can be gained over a comparable machine such as the IBM 704. The rest of this lecture introduces refinements which use the long instruction word to add to the power of a single order to an extent which, in practice, reduces the length of a program by a factor of two or three. Additional logical features further increase the flexibility of the commands. The reader is advised not to go beyond this point until he is fully familiar with the material to date.

CC AND PF

As already noted, CC contains the address of the next order to be executed at the end of any given order. The next order will normally start off by adding 1 to (CC). However, any transfer (HTR, TRA, SKP or JMP) will further modify (CC). In addition, on <u>unconditional transfers (Class 0)</u> only, (CC) -> PF before CC is modified a second time. Hence PF contains a record of the location <u>following</u> the one from which transfer was made. In other respects, PF acts as a normal B-register. Note that unconditional transfers also exist in class 4:

SCC XYZ is equivalent to TRA XYZ

ACC 1 is equivalent to SKP

But in the class 4 orders, (CC) does not go to PF.

Field I SETU

So far, it has been assumed that (U) remains undisturbed between orders, or is initially cleared by the CLA order. However, Field I allows any one of the 16 A and B series registers to be brought to U. When a 15-bit register is brought, it goes to U bits 40-54. Bits 1-6 are set to 0, and bits 7-39 are set equal to bit 40, i.e. bit 40 propagates to the left in $(U)_M$. Symbolically, if SETU is left blank, U is undisturbed (except by CLA), but any other special symbol in this field causes that register to come to U. After that, (U) may be further modified by a "-" or the absolute value "|" signs in Field I. To change the sign of (U), only the mantissa is complemented.

Examples (iv)

```
B4   ADD    B5     "Fixed point addition
                    (B4)+(B5)"
-|T7| MPY    PQ    "Fixed point multiply
                    -(PQ).|(T7)| "
-Z   AND    K      "Mantissa of (K) -> U"
```

Field IV INFLECTIONS

So far, 4 bits in Field IV are not accounted for. Two of these are used for sign inflection as in SETU, and are applied to (S) immediately before decoding the OPN field. Symbolically, an obvious notation is used. Another bit is used to indicate indirect addressing. In this, if the final address formed in Field IV is W, the number brought to S is not (W) but $((W)_{IV})$, assuming $(W)_{IV}$ does not contain an indirect addressing bit. Symbolically, a "*" causes indirect addressing to take place. As a final inflection in this field, the coder has the choice of bringing not (W) but the 15-bit number W to S. This is controlled by the "numerical" bit and is set symbolically by the lower case "a". By means of "a", unnecessary accesses to storage can be avoided.(1) It will be seen that decoding the ADDR+MOD field is a complex process. For full details, see the Computer Manual. For further details on symbolic forms, see the Note on AP1.

Example (v) Suppose $(00025)_{1-39}$=Z, $(00025)_{40-54}$=00061

Then

```
CLA   00025   brings 0000...61   to U
CLA  -00025   brings 0077 ...16  to U
CLA  *00025   brings (00061)     to U
CLA  a00025   brings   0000...25 to U
```

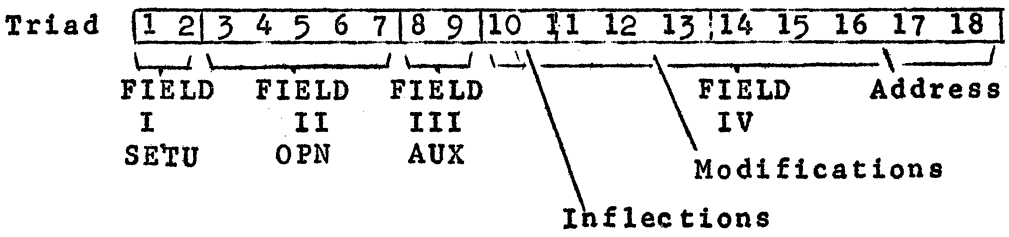
(1) This is sometimes called immediate addressing.

At the end of the operation, without loss in time, the coder can call for one of the following additions or transfers to take place. Field III AUX

Operation	Code	Example
Store U in some A or B register	U->A or B	U->T7
Store R in some A or B register	R->A or B	R->CC
Increment a B register by $\pm$ 1	Bi$\pm$1	B6+1
Increment a B register by $+(77772)_8$	Bi+X	PF+X
Send the final Field IV address from I to a B register	I->Bi	I->B3.

Note that in some cases AUX may effect a transfer of control. Symbolically, it is written after Field IV.

This completes the description of the instruction word. Its main sub-fields are summarized in the diagram:



Example (vi) We can rewrite Ex.(iii) as:

```
START      Z   SB1     d299,U->T5
               CLA     -a2
               STO     77772        (increment register)
           T5  FAD     XF+B1, U->T5
           B1  IF(PNZ)JMP Z, B1-1
           T5  STO     Y
```

Lecture 4. Subroutines, and some special registers.

SUB-ROUTINES

It can be seen that any arithmetic task and many simple logical tasks can be accomplished by a string of orders in symbolic or aboslute form. Such a string is called a routine or program. The routine uses pieces of data from storage and places its results back in storage and it is convenient to think of a routine for performing a specific task as a single logical unit. It is then called a subroutine. An example: The string of orders which takes (T7) and replaces it by $\sqrt{}$(T7) would be a "square root subroutine". This idea is the most important programming device.

Consider finding $y = \sqrt{x_1} + \sqrt{x_2} + \sqrt{x_3}$. We could write, assuming $L(x_1)$=X1, $L(x_2)$=X2, and $L(x_3)$=X3:

```
START        CLA   X1, U->T7
             △
       T7    STO   T6
             CLA   X2, U-> T7
             △
       T6    FAD   T7, U-> T6
             CLA   X3, U-> T7
             △
       T6    FAD   T7
             STO   Y
```

where △ stands for the N orders of the square root subroutine. Thus calculating y requires 3N+7 orders. Now suppose the order TRA PF is added to the end of △ to give △'. Then we could also write:

```
START       CLA   X1, U->T7
            TRA   SQRT
      T7    STO   T6
            CLA   X2, U->T7
            TRA   SQRT
      T6    FAD   T7, U->T6
            CLA   X3, U->T7
            TRA   SQRT
      T6    FAD   T7
            STO   Y
```

Δ' .

where SQRT is the symbolic location of the first order in Δ'. This method requires N + 11 orders to find y, so is more economical provided $N \geq 3$, which is the case. Δ is an example of an open subroutine. Δ' is closed. The latter are most commonly used.

LIBRARY SUB-ROUTINES

The commonest description of a computation is by means of a flow chart or block diagram. In it, a single "block" may stand for a subroutine which may itself be described by a flow chart, so the complete description is a recursive process, running to many flow charts, in which only the simplest can lead to direct coding. In mathematical problems, certain basic flow charts are common to many different calculations, and the whole process is simplified if the commonest functions are coded once and stored as a library of closed subroutines. These are stored on paper tape. Later, they will also be on magnetic tape.

The problem of communicating with library subroutines arises. Obviously, the less the coder has to remember about the subroutine the better. He should know what it does, and (analytically) how it works. He should know which fast registers it uses, and where it gets its data and where it stores its answers. But it should not matter where it is in storage or (in detail) how it is coded. Most of these requirements can be met by adopting certain conventions for library program usage, and later versions of the Assembly Program will further automate the communication between routines. The present conventions are:

CONVENTIONS FOR SUBROUTINES

(i) All library programs are completely "relativised" and may be translated to any part of the main store in single blocks.

(ii) They may use any fast registers, and the coder must first save those he doesn't want destroyed. Generally, a subroutine will use the T's in the order T4, T5, T6, T7; and the B's in the order PF, B6, B5, B4, ...B1.

(iii) Parameter values and arguments can be given in two ways:

(a) By placing ihem in the T's or B's (not U or R or S)

(b) By placing them in a "calling sequence" following the transfer order in the main routine, so that the subroutine can address them relative to PF; i.e., we have:

CALLING SEQUENCE

```
                    TRA   SUBR
      ((PF)):       (1st parameter)
    ((PF)+1):       (2nd parameter)
            --- etc. ---
  ((PF)+n-1):       (nth parameter)
```

The user must, of course, follow the choice of parameter storage made by the writer of the subroutine. Note that a parameter need not be a true argument, but may be the address of one, or the address of the address of one, etc., thus using the Indirect Addressing option.

(iv) Output values can be placed in T's, or B's, or in the calling sequence, or in the addresses given in the calling sequence, as in (iii).

(v) The exit order(s) from the subroutine always send control to the first word after the end of the calling sequence; i.e. (PF)+n for an n-word parameter list. If there is in fact a choice of K possible exits, then k, the exit parameter is stored in PF on leaving the subroutine. This gives the user the choice of acting on error conditions or special cases or ignoring them, or selecting the one he wants without lengthening the calling sequence.(1)

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

(vii) If a subroutine uses the Trapping, Sense or Mode registers (other than as input or output parameters) it must restore them before the exit order(s). The Indicator, X and PF2 registers must be saved (if necessary) by the user.

SPECIAL REGISTERS

The following remarks apply to the special 15-bit fast registers, 77770_8 through 77775_8. PF2(77770) receives (CC) before any transfer is made, other than normal sequencing. Its main use is in error diagnosis. SL(77771) is the Sense Light register, corresponding to 15 switches on the machine console. Each switch has 3 positions: ON,

(1) The writer of the subroutine can force the issue, if he thinks there are some conditions the user can never foresee and never ignore, by giving an extra return position in the calling sequence, but this gives rise to undesirable complications in the syntax of the coding language.

NEUTRAL, OFF. In the neutral position, it is under machine control, and can be turned on and off by the program (SLN, SLF). Otherwise, it is fixed in the ON or OFF position, and can only be interrogated by the program. To each switch corresponds a light, which is on when the switch is ON. The words "switch" and "light" are used synonymously. X(77772) is the increment register, used in Field III and JMP. ML(77773) is the mode light register, normally set to Z. The mode lights control the internal functioning of certain orders. It is possible, by turning the Repeat Mode light on, to suppress the normal advance of control and repeat the following order until some test is satisfied, or until a labelled number enters the Arithmetic unit, when the Repeat Mode light is turned off.

Example (i) To find the address (in B2) of the first non-zero word in memory.

```
        SB1   1
        ERM                              Enter Repeat Mode
    Z   IF(NZE)SKP Z+B1, I->B2   Is-(Z+(B1))=0?
```

Example (ii) To form the sum $\sum_{i=1}^{300} x_i$ (Lecture 3, Ex.(iii) and (vi)).

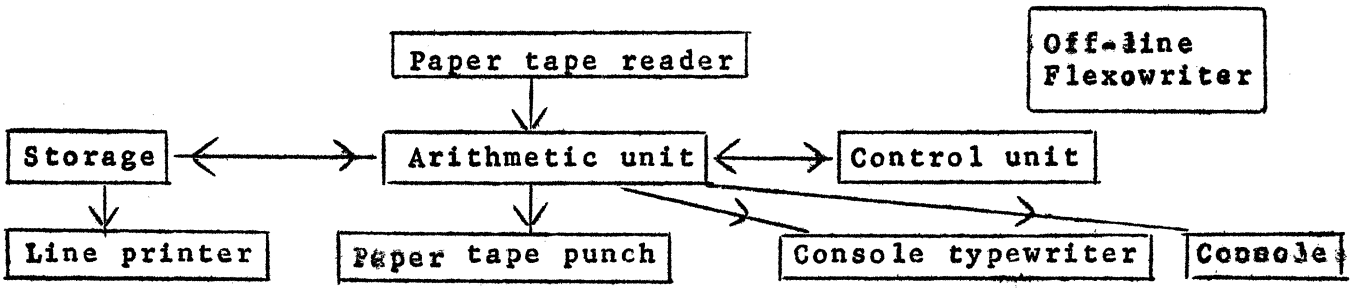
Assume that $L(x_i)$ = X1+i-1. Let (X1+d299) have Tag 1. Then the program is

```
        SB1   1
    Z   ERM   ,U->T5
   T5   FAD   X1+B1-1, U->T5
```

ARITH-
METIC
TAGS

The Trapping Register TR(77774) is not discussed here. It is ineffective provided the Trapping Mode Light is off. The Indicator Lights IL(77775) include exponent overflow, mantissa overflow, and arithmetic tag indicators no. 1, no.2, no.3. The latter are turned on and when a tagged word enters the arithmetic unit, they are tested and turned off by the class 0 orders (Lecture 3). The Computer Manual shows how tags can be placed on numbers in storage. Lecture 5 shows how tags can be placed on data by the input routines.

Lecture 5. Input and Output procedures.



INPUT AND OUTPUT

The diagram shows the input-output units currently included in the machine schematic. Note firstly that output via the line printer is independent of the arithmetic unit, so this interrupts computing only while the print order is initiated. Both the paper tape reader and punch disturb U, R, and S when they are used. Paper tape is prepared, reproduced, and transcribed to printed form by the off-line Flexowriter. It is the primary input medium, and the only permanent storage medium, but apart from this it is much inferior to the printer for putting out information: it is punched by the machine at around 30 characters per second, while the printer operates at up to 1000 c.p.s.

PAPER TAPE

A character on paper tape is represented by a six-hole code (except for 7-hole control characters, which are not read by the machine), which is directly translated into a six-bit Binary Coded Data (BCD) form by the machine. All other translations have to be programmed. For paper tape alone, subroutines are provided to convert from BCD form to binary numbers, and vice versa. Numbers are read in either in fixed or floating point decimal, or in octal form.

Each number is stored in one location, and numbers are delimited on tape by one of the separating characters (χ) , (comma), "tab" (tabulate), "cr" (carriage return).(1)

The primary subroutines are then:

(a) PDECIN: read one decimal word and leave it in binary form in T4. Floating point numbers are identified by a decimal point. All others are converted as fixed point integers and then shifted arithmetically. The number forms are:

DECIMAL INPUT

$\pm$ nn ... n.mm...mE $\pm$ ee T tχ (floating point)

$\pm$ nn ... n B $\pm$ ss T tχ (fixed point)

T indicates a tag t(=1,2, or 3); E a decimal power of 10; B a shift left (+) or right (-) which takes place after conversion. Floating point numbers are normalised. Fixed point numbers have Z exponent. Excessive digits are ignored, so a number should have less than 15 significant digits. Spaces are ignored.

Example (i) A typical data tape, with 3 floating point and 2 fixed point numbers, may be:

12., -.05E3, 1.026 E-7, 3B5, 0_{cr} .

The calling sequence for PDECIN is one order:

```
K      TRA      PDECIN
K+1    (point of return after executing the sub-
        routine)
```

Normally, on returning from the subroutine (PF)=0 and (T4) is the converted number. An error in the tape (an illegal character or a number our of range) will set (PF)=1 and (T4)=0.

(1) The octal codes for these are respectively 37, 22, and 24.

Example (ii) A program to read and store 5 numbers in (B1), (B1)+1, ...(B1)+4 is

```
        SB2     4
        TRA     PDECIN
T4      STO     Z+B1, B1+1
B2      IF(NZE)TRA CC-3, B2-1
```

(b) POCTIN: This, by analogy with PDECIN, reads in one octal number (fixed point form only, not more than 18 digits) and stores its binary equivalent in T4. An initial "-" sign on the number complements the final word after conversion and shifting.

(c) PDECEX: This is the inverse of (a). A number with Z exponent is converted as an integer. If non-zero exponent, the floating point form is

DECIMAL OUTPUT

$$\pm\ n.mm\ \ldots\ m\ E\ \pm\ ee\ T\ t$$

The number of places punched after the point is given by (B6), the last place being rounded in the usual decimal sense. Thus the inverse of Ex.(ii) is:

Example (iii). To punch 5 numbers in (B1), (B1)+1 ... (B1)+4 rounded to 3 decimal places:

```
        SB2     4
        SB6     3
        CLA     Z+B1, U->T4
        TRA     PDECEX, B1+1
T7      PCH     COMMA          (punch "," and space)
B2      IF(NZE)TRA CC-4, B2-1
```

where (COMMA) = 7437761400...0

and (T7) = 0000000100...0

The PCH order punches 4 rows of holes, which position the Flexowriter carriage in "upper case"; print ","; return to "lower case" and "space" one position.

Example (iv) The output from Ex.(iii) would read:

1.200E1, -5.00E1, 1.026E-7, 96, 0

(d) POCTEX: This is the inverse of (b). 18 octal digits are punched, except for leading zeros.

(e) Direct transcription of BCD codes between memory and tape is effected by the RDH (read hexad) and PCH (punch hexad) orders. To punch (or read) 9 characters, a 1 is brought to bit 54 of U, and the source (or destination) is given by field IV.

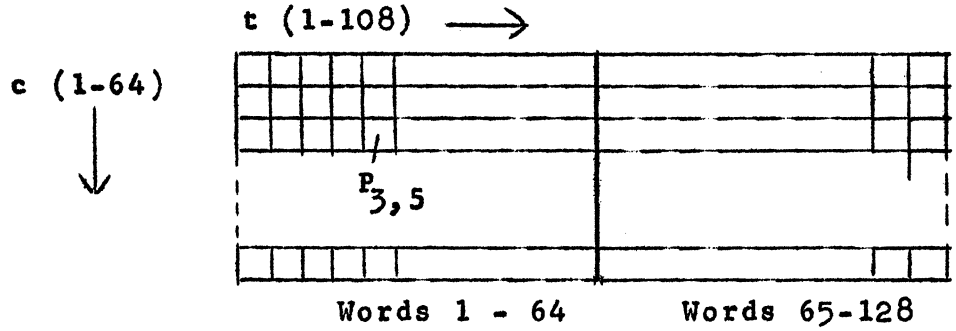
Example (v). Let (B1)=1.

```
Then    B1    RDH   1017     reads 9 characters
                               into 1017
        B1    PCH   SIGMA    punches 9 characters
                               from (SIGMA).
```

Both (R) and (U) are destroyed by these orders.

PRINTER

The printer is for output only. The Print Matrix is the name of the 128 consecutive memory locations reserved for the printing of any <u>one</u> line, and for which one "print" order is necessary. A 1 at position $P_{3,5}$ causes the printing of character no. 3 at position 5 on the printed sheet.



The choice of the first word of the print matrix is arbitrary.

Example. To print in positions: 1 2 3 4 5 6 7...106 107 108

The characters: B A F D C A

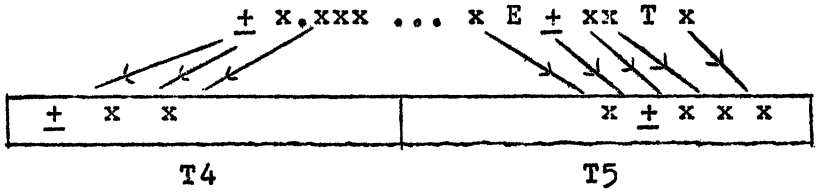
we form:

c	t	1	2	3	4	5	6	7	...	54	55	56	...	106	107	108
	A	0	0	0	1	0	0							0	0	1
	B	0	1	0	0	0	0							0	0	0
	C	0	0	0	0	0	0							0	1	0
	D	0	0	0	0	0	1							0	0	0
	E	0	0	0	0	0	0							0	0	0
	F	0	0	0	0	1	0							0	0	0
		Words 1 - 64									Words 65 - 128					

Formats give the line spacing pattern to be followed; Class 5 orders specify address of first word in matrix and the format to be used.

The basic subroutines for the printer assist in preparing a print matrix from words coded in the six-bit characters corresponding to the position of a symbol on the print wheel. In the case of certain symbols, including the signs (+ -) and numbers, these codes are identical with the BCD form.

(f) BINDEC converts one number (T4) from binary to BCD form, leaving the results in the T4 and T5. The above floating point decimal format can conveniently be compressed to 18 characters by omitting ".", "E", and "T":



The calling sequence is:

```
K:    TRA   BINDEC
K+1   (normal return)
```

with two exit parameters 0 = floating point no., 1 = fixed point number. A fixed point number is left justified and terminated by a "34" octal code..

Thus 13 significant decimal digits are available. They can be punched out as Flexowriter codes, or converted to the print matrix form by:

(g) PRINTR which takes six-bit printer codes from a given address and stores them in a given position of the print matrix, which is also specified. A single parameter contains in packed form all the information required:

```
K:    TRA  PRINTR
K+1:  (Pack) FWA(1) of Matrix (15 bits), Data Loca
      (15 bits), No. of symbols (9 bits), Type
      position (9 bits).
K+2:  (Return address)
```

Characters are taken from the left side of the Data Location. Only 9 or fewer characters can be specified. In case the six-bit printer codes and the BCD codes are not the same, a more elaborate conversion routine is necessary. However, (f) and (g) are adequate for numerical output. The final matrix is printed with a single "PR1" order (print and space one line).

---

(1) Commonly "FWA" stands for "First word address". Similarly, "LWA" for "Last word address".

Example (vi) To print out (W), (a floating point number) with five places of decimals (assuming no tag) in printer positions 9-20.

```
        CLA   W, U->T4
        TRA   BINDEC
T4      LUR   d12, U->R          (truncate)
Z       LLS   d24
        LUL   d6
        ORU   a33              (insert decimal point)
        LRS   d18,R->T4
        TRA   PRINTR           (mantissa)
        OCT   0100000004010011 00
T5      LUR   6
        LRS   d18, R->T4
        TRA   PRINTR           (exponent)
        OCT   0100000004003022 00
        PR1   01000            (print)
```

The subroutine library will itself contain more elaborate routines of the type in Ex.(vi), but (a) - (g) are given as fundamental building blocks, to which the coder can add with the help of the manual.

The console typewriter can be used for direct printed output, at about 10 c.p.s. It is controlled either manually, or by the program; and is mainly used for checking purposes, as discussed in the next lecture, together with the use of the console controls.

Lecture 6. Miscellaneous comments on using the machine.

There is no "operating system" for the machine, and the coder is free to devise his own procedures at every stage of problem solving. The mechanical details are simple but with the option of using a number of coding aids, he can hand over an increasing proportion of the detailed routine work to the machine. The five stages of solution from problem analysis onwards are:

(i) Preparation of a logical flow chart
(ii) Preparation of machine codes in absolute or API form
(iii) Program assembly
(iv) Program testing
(v) Program execution

and the main aids to coding are under headings (iii) and (iv).

ASSEMBLY PROGRAM

API is a program of about 1000 orders, and is read into the machine immediately prior to feeding in the symbolic paper tape. It is technically a two-pass system, and in the absence of magnetic tape units the tape with symbolic codes must be read into the machine twice. In pass 1, fields I, II, and III are decoded, B-modifiers and inflections are stored and a Symbol Table is formed associating an absolute address with each Symbol that appears. Since a symbol need not be defined until after its first appearance, a second pass is necessary to fill in all the correct addresses in Field IV. Various output options are provided, and are controlled by Sense switches. The final machine

program can be punched out in a condensed binary form, and it is automatically headed by a loading routine and followed by a "stop" code, so in order to use it the programmer has only to put the tape back into the machine and press the "load" key on the console, and then transfer to the starting address. API tapes are not relocatable, but with little extra effort the coder can write "relativised" code where all transfers are made relative to (CC), and this amounts to the same thing.

PSEUDO ORDERS

The pseudo-orders given for API are a means of controlling the assembly program itself, thus affecting the final program only indirectly. They are not included in the final output tape. They allow for input of numerical and BCD data at assembly time, for the control of printed output, and the allocation of symbols and storage. In particular, it should be noted that a whole program can be written as a string of octal numbers preceded by the OCT order, and this is the form in which Library subroutines are written.

LIBRARY ROUTINES

Library routines can be read in at any point in the program. They should be identified by the mnemonic code given in the program abstract, either directly or via an EQU order. No symbols in the main program, other than the subroutine name, have correspondence with any location in the subroutine.

The attached example gives a short program in symbolic form, its final absolute form and associated symbol table.

At the end of the first pass the symbol table can be obtained, punched in binary form. It contains enough information to allow additions to be made to the program at a later stage without complete reassembly.

After assembly is complete, an attempt must be made to run the program which has been produced. The control console provides a number of keys for actuating the machine, and lights for observing its status at any time. The main keys are:

1. CLEAR: which sets all 56 bits of each storage word to 0.
2. LOAD: which causes a RDH order to be placed in I, and actuates the paper tape reader to place the first word in 00010_8.
3. STOP: which stops the machine at the end of the current order.
4. START: which sends control to (CC) for the next order.

The main lights are:

5. Control Counter CC (15 bits)
6. Sense lights SL (15 bits)
7. Indicator lights IL (15 bits)
8. Instruction register I (54 bits)

Adjacent to or on the console are the line printer, paper tape reader and punch, and the console typewriter, which acts as an input and output device to all the fast A and B series registers and the special 15 bit fast stores (SL, IL, etc.). It is possible to type octally into or out of any of these locations, and to obtain a printed record of all pieces of information transferred. A convenient way of starting from a particular location is:

TYPEWRITER

(a) Enter the location number in CC via the typewriter
(b) Press START.

Any number of events may cause the machine to stop. Some may be important, and others the coder may want to ignore for the time being. There is an "Ignore error stop" switch which causes invalid orders to be bypassed when it is set to ON. In addition, a stop may be caused by any of the following situations:

STOPS

(a) A programmed HTR or a satisfied conditional IF(XXX)HTR
(b) A "zero" order, which is effectively a HTR
(c) A 2-bit parity failure in memory.
(d) A divide check
(e) An input-output order waiting for the appropriate unit to be switched ON.
(f) Depressing the STOP key.

Generally, the failure of a program to produce a desired result can be attributed to any one of the four main error sources:

ERRORS

(a) Machine faults such as parity failures.(1)
(b) Typographical errors in data or program tapes.
(c) Arithmetic errors in the program.
(d) Logical errors in the program.

A full analysis of the detection and diagnosis of these would constitute a lengthy treatise, but by observing some general rules the coder may aim at minimising the occurrence of errors under each heading.

READ-AROUND ERROR

(a) Too little is known of the machine at the moment to be specific on this point, but the nature of electrostatic storage places a slight restriction on coding. It is

(1) i.e., an error in two or more bits in a word in storage. Single bit errors are corrected by keeping a number of "parity" check bits in addition to the "working" information.

known that sensing a storage spot will disturb adjacent spots on the surface of a storage tube, and the cumulative effect of many read or write operations in one spot can induce errors nearby. The upper limit to the number of operations is between 250 and 500. However, each spot is automatically regenerated about every 60 milliseconds, which is the time taken to execute about 2000 orders. Hence tight loops of 8 or less orders which may be executed more than 250 times should be used with care. In critical cases, the use of T-registers for orders or data, or the use of the Repeat Mode, may avoid danger.

(b) The main purpose of AP1, and the data input routines, is to allow orders and numbers to be written in a natural and easily recognised way. During assembly, apparent errors will cause a printed comment on the output listing. In doubtful cases, a NOP order is inserted in the program by AP1.

(c) and (d). At present, no help is given in detecting these types of errors prior to execution. The coder must run his program, and then see where it breaks down. The most useful devices in detecting the source of an error are frequent print-outs of intermediate results (which may be controlled by Sense Light tests in the program, and suppressed when it is working correctly), "Dump" routines (which achieve the same effect by modifying the program just before execution),

and "Trace" routines, which print out the contents of the arithmetic registers during the execution of a particular sequence of orders. The coder will add his own techniques to this list. The main point is that errors always occur, and the error detection techniques should be devised while the code is being written. Standard Trace and Dump routines are part of the program Library.

One final point. A machine of this type is strongly oriented towards "closed subroutine" programming: relativisation is easy, and communications between routines are almost automatic. Any coder is urged to take advantage of this, and divide his program into distinct, relativised, closed subroutines operating, possibly, on a fixed common data region. The reason is that check-out time is not a linear function of the length of a program, and it is most efficient to get many short programs working separately and then fit them together to form a long one. Later versions of the Assembly Program emphasise this technique.

An example of API and absolute coding.

The problem is to calculate the surface area of the sphere centered at the origin, with a radius vector given by (x, y, z) where L(x) = (B1) and y and z are stored in the two following locations, and to store the result in location A.

The appearance of symbolic codes on the coding sheet is:

LOCATION	SET U	OPERATION	ADDRESS + MOD, AUX	REMARKS
		ORG	1025	
BEGIN	Z	SB2	2, U-> T5	
		CLA	B1 + B2 + Z	
		FMP	U	
		FAD	T5, U -> T5	
	B2	IF (NZE) TRA	CC - 4, B2 - 1	
	T5	FMP	FOURPI	
		STO	A	
FOURPI		DEC	12.5663706	
A		EQU	d566	
		END		

The output of the assembly program is:

Location	Instruction
01025	00 44200 05 4000 00002
01026	00 50000 00 0006 00000
01027	01 10600 00 0000 00001
01030	01 10400 05 0000 00005
01031	42 05050 62 4001 77773
01032	05 10600 00 0000 01034
01033	01 20000 00 0000 01066
01034	01 14441 76 6517 10652

Note that there is an exact correspondence between each executable order and the absolute codes; use of API does not affect execution time.