TX-0 COMPUTER
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
CAMBRIDGE 39, MASSACHUSETTS

M-5001-19-1

August 23, 1961

SOME USEFUL MICRO- AND MACRO-INSTRUCTIONS

The instruction vocabulary of MACRO III can be enriched considerably through parameter assignments and macro-instruction definitions. This memorandum is intended to help persons who are relatively new to TX-0 to acquaint themselves with some of its unusual properties (and peculiarities), particularly the microprogramming feature.

Many of the instructions discussed below are not currently defined in MACRO III. However, a tape containing most of these definitions is available from the subroutine library as an English tape (ENG DEFS A) which may be copied into the beginning of the users' English tape. This set of definitions is also available as a binary symbol tape (BIN DEFS A) which may be used to restore MACRO before making a conversion. The content of these tapes is given on the last page of this memorandum.

1. Operate Instructions

The following operate instructions not currently defined in MACRO III are frequently useful. Some of these operations are discussed below in more detail.

Instruction	Operation
ill = lal+alr-opr	$IR+LR \rightarrow AC$, $AC \rightarrow LR$ (equivalent to ial followed by cyl)
anc = ana 40	$-(AC \cap LR) \rightarrow AC$ (ana followed by com)
anz = ana 50	$(AC \cap LR) + (-0) \rightarrow AC$ (anc followed by amz)
orc = ora 40	$-(AC \cup LR) \rightarrow AC$ (ora followed by com)
pnl = prt+ cyl-opr	(prt followed a cyl)
ran = cyr+ cry-opr	(random number generator) see below

In the TX-0 (and many other machines), addition is accomplished in two steps: partial add and carry. These two operations are also separately useful as operate commands.

1.1 Partial Add

The partial sum of two numbers ($x \wedge y$) is a number each of whose bits represents the "exclusive or" between corresponding bits of x and y. In terms of the Boolean functions "and" ($\cap$), "or" ($\cup$) and "not" (-), then,

$$x \wedge y = (x \cap - y) \cup (y \cap - x). \tag{1}$$

In particular, it can be seen that

$$x \wedge 0 = x \tag{2}$$

$$x \wedge x = 0 \tag{3}$$

$$x \wedge -0 = -x \tag{4}$$

Since this operation is associative, it follows from (2) and (3) that

$$x \wedge y \wedge y = x \tag{5}$$

The command lac (LR→AC) makes use of relation (2), while lcc (-LR→AC) uses (4). lpd (LR $\wedge$ AC→AC) can be used to advantage in a number of ways. For example, suppose that it is desired to combine bits 0 - 8 of the AC and bits 9 - 17 of the LR into a single word (the unwanted portions of both registers <u>not</u> assumed to be clear). Making use of (2) and (5), the desired result can be obtained in two steps:

```
lpd
clr+lpd-opr
```

1.2 Carry

The carry operation is loosely analogous to the carry used in pencil-and-paper addition. It may be defined as follows:

> A carry digit is a 1 if, in the next less significant bit, either AC = 0 and MBR = 1, or AC = 1 and the carry digit = 1. The carry digits so determined are partial added to the AC.

Let the result of a carry operation be denoted by AC $\overset{c}{-}$ MBR since (in the one's complement arithmetic of TX-0) the addition operation is the result of the successive execution of partial add and carry, we may write

$$x + y = (x \wedge y) \overset{c}{-} y \qquad (6)$$

If we define

$$z = x \wedge y$$

it follows from (5) that

$$z \wedge y = x \wedge y \wedge y = x$$

Hence (6) may be written

$$z \overset{c}{-} y = (z \wedge y) + y \qquad (7)$$

In particular, it may be noted that

$$0 \overset{c}{-} y = (0 \wedge y) + y = 2y \qquad (8)$$

which gives rise to the lal command (LR + LR→AC), and that

$$-y \overset{c}{-} y = (-y \wedge y) + y = -0 + y, \qquad (9)$$

which is represented by amz (-0 + AC→AC).

The instruction cry (AC $\overset{c}{-}$ LR→AC) can be used in a number of ways. For example, suppose that it is desired to sense whether bit 9 of the LR contains a 1 or 0. This can be accomplished as follows:

```
cla
add (777377
cry
trn ha
```

It can be seen that if bit 9 of the LR were 0, cry would not change the AC, so control would be transferred to ha. If bit 9 were 1, however, the entire AC would be complemented and the transfer would not be made.

As a second example, suppose that it is desired to cycle left the contents of the LR treating the 12 least significant bits as a unit (i.e., bypassing bits 0 - 5). This is accomplished by

```
cla
add (770000
cry
llr (7777
ano
```

Following this set of operations, the AC and bits 0 - 5 of the LR will be clear, while bits 6 - 17 will be cycled left one position (the former bit 6 going to bit 17).

The instruction ran (= cyr+cry-opr), when used with an appropriate number in the live register, can be used to produce a sequence of psuedo-random numbers. In particular, the instructions

```
cla
add t
llr (355670
ran
sto t
```

can be used repeatedly to generate a sequence of approximately 250,000 random 18-bit words in register t (eventually, a previously computed value is obtained and the sequence repeats). Plus zero is a good choice for the initial contents of t. Minus zero should not be used.

To illustrate another application of ran, suppose that it is desired to make a four-way branch (to locations b0 through b3) dependent on bits 9 and 10 of the LR. Using the same principle that was used before, this is accomplished by

```
cla
add (777377
cry
trn .+4
ran+com-opr
trn b1
tra b0
ran
trn b3
tra b2 .
```

2. Macro-instructions

The following instruction sequences occur often enough in many programs to warrant their definition as macros.

MACRO	INSTRUCTION SEQUENCE
clad A	cla add A
llac A	llr A lac
llcc A	llr A lcc
move A,B	llr A slr B
load A,C	llr (C slr A
acst C,A	add (C sto A
step A,C	cla add (C add A sto A

test C,S	add (-C trn S
call S	llr (tra .+2 tra S
subr S,T	llr (tra T tra S

3. Repetitions

An aspect of programming for the present configuration of TX-0 that differs from many computers is the frequent need for repetition of instructions (e.g. in cycling and shifting). The psuedo-instruction repeat may be used for this purpose. Inserting

```
repeat n, ins
```

into a program causes the instruction ins to be assembled n successive times in the binary program. For example, if the programmer wishes to cycle the AC nine places to the right, he can write

```
repeat 9, cyr .
```

A more detailed description of this psuedo-instruction is given in Preliminary Operation Instructions for MACRO III (M-5001-35).

In some cases, the programmer may wish to specify the number of repetitions at some later time by means of a parameter assignment. For example, if he wishes to cycle left zip times ($0 \leq zip \leq 8$), he may write

```
repeat zip, cyr
```

If a large amount of shifting or cycling is required, it is generally more efficient to use closed subroutines together with macros, such as the following:

```
            zz=.
            define      shrc C      llr (tra .+2
                                    tra zz+22-C         terminate
                                    repeat 18, shr
                                    slr .+1
                                    0
```

The programmer can then shift right zip places by writing

```
                                    shrc zip
```

where zip could be any combination of constants or parameters whose sum lies in the interval $(0, 18_{10})$.

In some cases it may be desirable to repeat some operation a variable number of times, depending on the result of an earlier computation. This can be accomplished in a similar way. For example, if it were desired to multiply the contents of the LR by the AC, knowing that $0 \leq AC \leq 7$, the following sequence could be used:

```
                                    com
                                    add (tra .+13
                                    sto .+2
                                    cla
                                    0
                                    repeat 7, lad
```

In a similar way, a variable number of shifts or cycles can be obtained by defining appropriate calling sequences as macros and using the same kind of closed subroutine as the one above under shrc C.

4. Remarks

Undoubtedly, other generally useful instructions or instruction sequences have been developed by TX-0 users. It is recommended that these techniques be documented (for the good of the cause and the barishment of dis-organized labor).

Hopefully, many of the instructions discussed above will be made obsolete by the more powerful set of hardware commands that is planned for the TX-0. The full potential of the new commands can best be realized through communication between the users.

```
|DEFINE A

ill=lal+alr-opr    pnl=prt+cyl-opr    anc=ana 40
anz=ana 50         orc=ora 40         xx=hlt
ran=cyr+cry-opr

define  addc C     add (C                      terminate

define  clad A     cla          add A          terminate

define  clac C     cla          add (C         terminate

define  acst C,A   add (C       sto A          terminate

define  step A,C   cla          add (C
                   add A        sto A          terminate

define  llrc C     llr (C                      terminate

define  llac A     llr A        lac            terminate

define  llcc A     llr A        lcc            terminate

define  move A,B   llr A        slr B          terminate

define  load A,C   llr (C       slr A          terminate

define  test C,S   add (-C      trn S          terminate

define  subr S,T   llr (tra T
                   tra S                       terminate

define  call S     subr S,.+2                  terminate

start
```