PROGRAMMING & SIMULATION DEPARTMENT
Sylvania Electronic Systems
A Division of Sylvania Electric Products Inc.
Data Systems Operations
Needham, Massachusetts

PRELIMINARY NOTES - MOBIDIC PROGRAMMING COURSE

These notes are based on the material covered at Fort Monmouth during the two 2-week MOBIDIC programming courses given from October 13, 1958 - November 7, 1958. This material is considered only in preliminary form and should be examined in that light. The notes are not meant to be self contained and must be used in conjunction with the Preliminary Programming Manual and the appropriate lectures.

The notes are being issued at this time in their unpolished version because it is felt that they will be of value now, even though in preliminary form. Some errors are known to exist on the typed sheets. A final version of the same material is expected to be made available at some later date.

Copy No. 35R1

December 1958

OUTLINE FOR PRELIMINARY NOTES - MOBIDIC PROGRAMMING COURSE

This page may be used for special notes if desired.

PRELIMINARY NOTES

MOBIDIC PROGRAMMING COURSE

1. NUMBER SYSTEMS

1.1 Examination of Number Systems

1.1.1 Decimal Numbers are numbers having a radix or base of 10.

Example: $345 = 300 + 40 + 5$

$= 3 \times 10^2 + 4 \times 10^1 + 5 \times 10^0$

1.1.2 Binary Numbers are numbers having a radix or base of 2. The only digits are 0, 1.

Example: $(1011)_2 = (1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0)_{10}$

This may be written either as $(1011)_2$ or $(11)_{10}$.

One binary digit = one bit. (Binary bit is redundant).

The MOBIDIC operates in the binary mode. This system has the disadvantage of using many more digits to express a value than the decimal number system; therefore, to save space and time, programming will be done in the octal number system.

1.1.3 Octal Numbers are numbers having a radix or base of 8. Only digits 0 ... 7 exist in this system.

Example: $(246)_8 = (2 \times 8^2 + 4 \times 8^1 + 6 \times 8^0)_{10}$

$(246)_8 = (166)_{10}$

The relationship between binary and octal is extremely convenient because 8 happens to be a

power of 2. Since it is the third power of 2, binary numbers are taken in groups of three for conversion to octal numbers.

The binary point is used in the binary system, decimal point in the decimal system, octal point in the octal system, etc. A general term covering the point in any system is the Radix point.

In general, $N = a_n b^n + a_{n-1} b^{n-1} + a_1 b^1 + a_0 b^0$

where b is called the base.

For a fractional number,

$N = \ldots a_{-1} b^{-1} + a_{-2} b^{-2} + \ldots$

1.2 Conversion from One Base to Another

1.2.1 Decimal to Binary or Octal

1.2.1.1 For integers, use a successive division by the base; the remainders will give the digits.

Example: Convert $(25)_{10}$ to binary

2 | 25	Remainder	1	
2 | 12	"	0	
2 | 6	"	0	
2 | 3	"	1	
2 | 1	"	1	(most significant digit)

$(11001)_2 = (25)_{10}$

Example: Convert $(1809)_{10}$ to octal

8 | 1809	Remainder	1
8 | 226	"	2
8 | 28	"	4
8 | 3	"	3

$$(1809)_{10} = (3421)_8$$

1.2.1.2 For fractions, multiply the fractional part by the base and record the integral part of the results. The digits, from the top down, give the numbers in the new base.

Example: Convert $(.62)_{10}$ to binary

.62	
1.24	1 (most significant digit)
.48	0
.96	0
1.92	1
1.84	1
1.68	1
etc.	

$$(.62)_{10} = (.100111)2$$

Notice that this is not exact because fractions in one base are not usually finite in another base, e.g.,

$$1/3 = (.333...3)_{10} = (.1)_3$$

1.2.2 Octal to and from Binary

Conversion from binary to octal is done merely by interpreting the binary digits in groups of three.

Example: $(10\ 111\ 001\ 011)_2 = (2713)_8$

$$(4702)_8 = (100\ 111\ 000\ 010)_2$$

1.2.3 Binary or Octal to Decimal

Examples: $(1011)_2 = (1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0)_{10}$

$$(246)_8 = (2 \times 8^2 + 4 \times 8^1 + 6 \times 8^0)_{10}$$

1.2.4 Mixed Decimal to Octal or Binary

Converting a decimal number which is part integral and part fractional can be done in three ways:

1.2.4.1 Convert each part separately.

Example: $91.42 = 91 + .42$

$$= (1011011)_2 + (.0110110)_2$$

$$= (1011011.0110110)_2$$

1.2.4.2 Convert the entire number as an integer and then multiply the binary result by the binary equivalent of the necessary power of 10.

Example: $91.42 = (9142) \times 10^{-2}$

$$= (10001110110110)_2\ (.00000010100011)_2$$

$$= (1011011.0110110)_2$$

1.2.4.3 Convert the entire number as a fraction and then multiply the binary result by the required power of 10.

Example: $91.42 = (.9142) \times 10^2$

$$= (1100100)_2\ (.11101010000010)_2$$

$$= (1011011.0110110)_2$$

1.3 Binary Arithmetic

1.3.1 Addition

0 + 0 = 0

0 + 1 = 1

1 + 1 = 10

1.3.2 Multiplication

0 x 0 = 0

0 x 1 = 0

1 x 1 = 1

1.3.3 Subtraction

1.3.3.1 First Method - Subtract smaller from larger and use sign of larger.

Examples:	1011	1100101
	-0101	-1010110
	110	0001111

1.3.3.2 Second Method - This utilizes the 2's complement of the number being subtracted.

The 1's complement of a number is found by subtracting every digit from the number which is one less than the base of the number system. In binary, we subtract each digit from 1. (It turns out that this is equivalent to interchanging 0's with 1's and 1's with 0's). In the decimal system we would subtract every digit from 9.

The 2's complement is found by adding 1 to the 1's complement.

Rules for subtraction:

a) Treat numbers as fractions.

b) Take the 2's complement of the subtrahend and add to the minuend.

c) If the answer is larger than or equal to 1, the fractional portion of the answer is correct.

d) If the answer is less than 1, take the 2's complement of the result and change the sign.

Decimal example 1:

493201 - 126944 = x

2's complement of 126944 = 873056

.493201

+ .873056

1.366257 x = 366257

Result is greater than 1, so correct answer is .366257.

Decimal example 2:

493201 - 602356 = x

2's complement of 602356 = 397644

.493201

+ .397644

.890845

2's complement of 890845 = 109155; x = - 109155

Binary example 1:

1110 - 1010 = x

2's complement of 1010 = 0101 + 1 = 0110

```
  1110
+ 0110
-------
 1.0100        x = 0100
```

Result is greater than 1 so answer is 0100.

Binary example 2:

1110 - 1111 = x

2's complement of 1111 = 0001

```
  1110
+ 0001
-------
  1111
```

2's complement of 1111 = 0001

x = - 0001

1.3.4 Division

In computers, division is done by a series of successive subtractions.

Example:

```
        1001
    ------------
1011 | 1101101
       1011
       -----
        10101
         1011
         ----
         1010   remainder
```

1.4 Sample Problems in Conversion and Arithmetic

Decimal ----→ Binary

$(149)_{10}$ =

2) 149	
74	1
37	0
18	1
9	0
4	1
2	0
1	0
	1

$= (10010101)_2$

$(459)_{10}$ =

2) 459	
229	1
114	1
57	0
28	1
14	0
7	0
3	1
1	1
	1

$= (111001011)_2$

$(.149)_{10}$ =

149	
2	
298	0
596	0
192	1
384	0
768	0
536	1
072	1
144	0
	0

$= (.001001100)_2$

Decimal ----→ Octal

$(923)_{10}$ =

932	
116	4
14	4
1	6
	1

$= (1644)_8$

$(1673)_{10}$ =

1632	
209	1
26	1
3	2
	3

$= (3211)_8$

Octal ----> Binary

$(457)_8$ = $(100\ 101\ 111)_2$

$(1632)_8$ = $(001\ 110\ 011\ 010)_2$

Octal ----> Decimal

$(777)_8$ = 448 + 56 + 7 = $(511)_{10}$

$(410)_8$ = 256 + 8 + 0 = $(264)_{10}$

Binary ----> Decimal

256	128	64	32	16	8	4	2	1	<----- Power of two
(1	0	0	1	1	0	1	0	$1)_2$ = 256 + 32 + 16 + 4 + 1 = $(309)_{10}$	1101100110 = 512 + 256 + 64 + 32 + 4 + 2 = $(870)_{10}$

ADD:

```
  1010101        101
  1101010        110
1 0111111         11
                   1
                 100
                  10
                10101
```

SUB:

```
  11010
  10111
  00011
```

MULT:

```
    101011
     11101
    101011
   1010110
  101011
 101011
100110111111
```

DIV:

```
          10001
10111) 11001101
       10111
       00010101
```

2. COMPUTER ORGANIZATION

2.1 Basic Computer Organization

2.1.1 Place of Digital Computers in Modern Society

a) They permit the solution of scientific problems which could not be solved previously.

b) They are used in business systems because they are able to obtain accurate results at greater speed than humans.

2.1.2 Functions of Digital Computers

a) They perform arithmetic operations rapidly.

b) They make simple decisions.

c) They follow instructions explicitly, even those which are wrong. They are in no sense an electronic brain.

d) Their greatest asset is speed.

2.1.3 Components of Digital Computer (See Figure 1)

a) Arithmetic Unit

b) Storage Unit

c) Control Unit

d) Input and Output Units

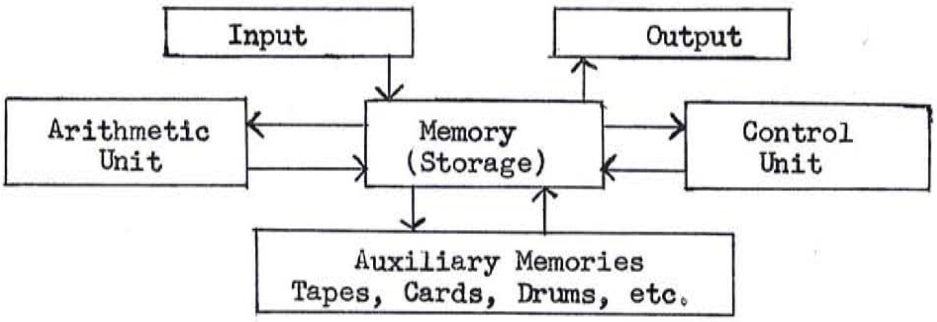


Figure 1

2.1.4 Basic Concepts of Present Digital Computers

a) A "stored program" computer is one whose program can be stored before calculation begins.

b) Data and orders are stored in the same unit and in the same manner.

Orders may be operated on as numbers.

2.1.5 Types of Orders

a) Arithmetic Orders: add, subtract, shifts, etc.

b) Data Transfer Orders: load, move, store, etc.

c) Control Transfers: conditional and unconditional.

d) Input and Output Orders.

2.2 MOBIDIC COMPUTER ORGANIZATION

2.2.1 Design

The MOBIDIC was designed for mobile installation but can be used in fixed location equally as well.

2.2.2 Unit Composition

The MOBIDIC System is composed of the following machine units:

Central Computer

Memory Units

Input-Output Converters

Flexowriter Output Units

Paper-Tape Readers

Paper-Tape Punches

Magnetic Tape Units

Communications Equipment

Power Supplies and Air Conditioning Units

Control Console

The actual number of machine units used in each case is variable, and depends on the particular application under consideration.

2.2.3 Interconnection of Machine Units

Interconnections of these units are shown in Figure 2.

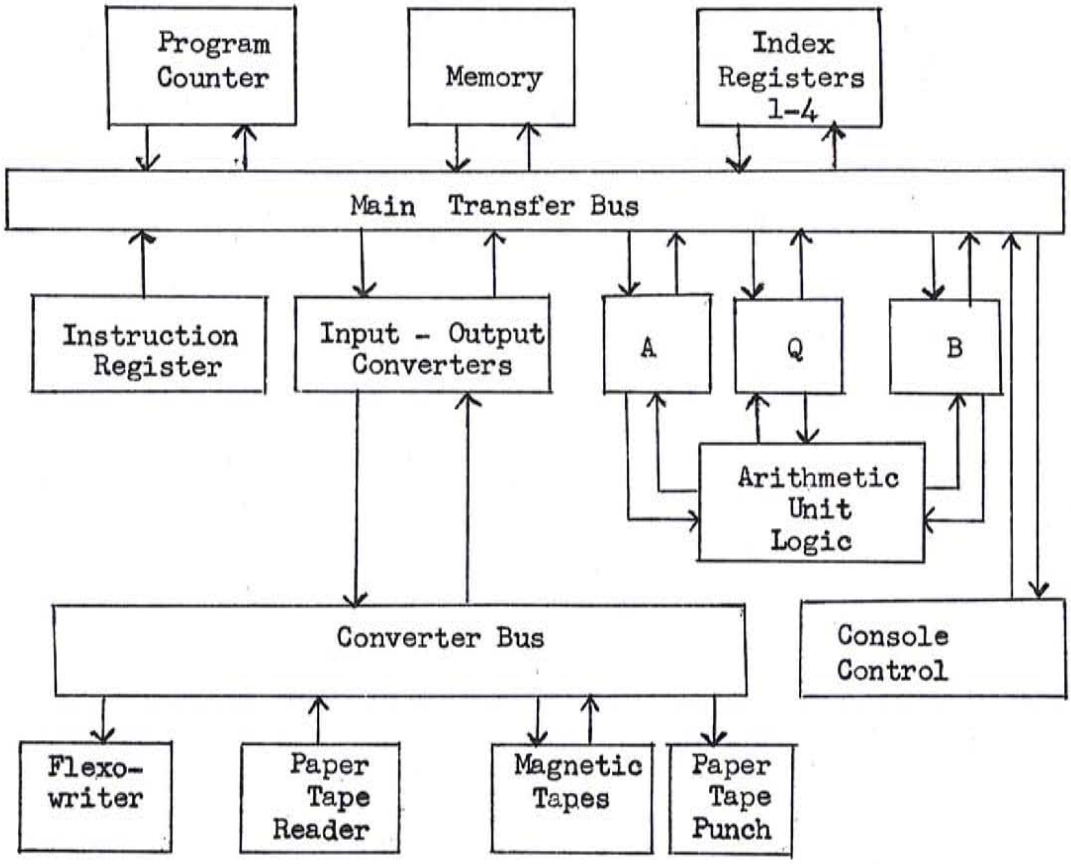


Figure 2

2.2.4 Particular Features of the MOBIDIC

a) Input - output orders may proceed independently of the computation.

b) Input - Output may proceed in real-time; i.e., "asynchronously".

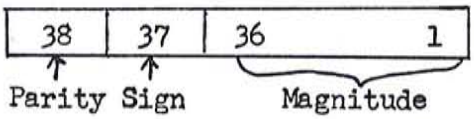
c) The machine can be expanded to contain 4 in-out converters, 7 memory units (each containing 4096 words), 7 index registers and 63 in-out devices.

2.2.5 Word Format

Word Format is 38 bits: bits 1-36 represent the magnitude of the instruction, bit 37 is the sign (0 = plus, 1 = minus), and the 38th bit is used as a parity bit, i.e. it is chosen by the machine so that the total number of binary ones in the word is odd. There are two types of words, data words and orders. Each of these is further subdivided into two forms:

2.2.5.1 Data words:

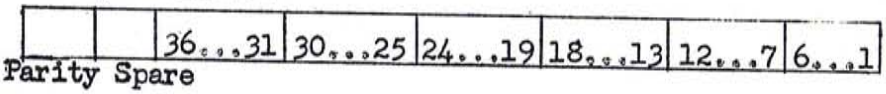
Numeric:



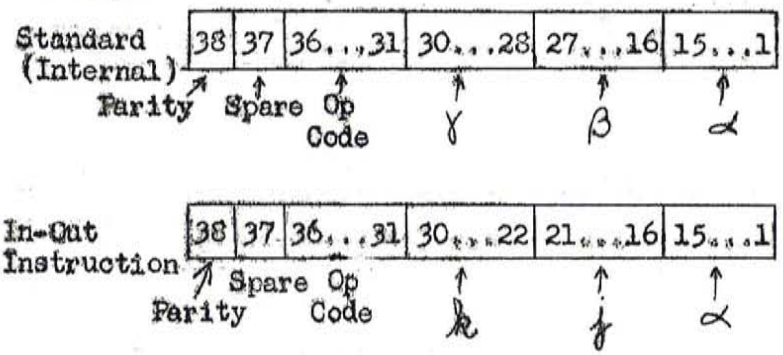
Binary point is between bits 36 and 37.

Note: Negative numbers are stored as absolute values with sign.

Alphanumeric:



2.2.5.2 Orders



Standard Instruction Word:

Alpha (α) denotes address to be used. Bits 13-15 of the address specify which of seven possible memories is to be used (addresses from 0 - 67777_8 are legal). If bit positions 13-15 are all 1, an addressable register is specified.

Beta (β) can be used

a) As part of a second address

b) As a constant

c) To control trapping mode

d) To control overflow

Gamma (γ) can be used either to specify index registers or as part of a second address.

The Op Code, bits 31-36, denotes the operation to be performed.

Input-Output Instruction Word:

Alpha, (α) bits 1-15, denotes the address.

J, bits 16-21, specifies the in-out device.

K, bits 22-30, specifies the amount of information to be processed.

Op Code, bits 31-36, denotes the operation to be performed.

2.2.6 Addressable Registers of Primary Importance

2.2.6.1 The A Register, or Accumulator, consists of 37 bits, and is the main register in the arithmetic unit. Most arithmetic and comparison operations involve the contents of the accumulator. The accumulator may be reset to zero before entering new information, or new information may be added to the contents of the accumulator without prior resetting. The accumulator holds the sum, difference, product and remainder after addition, subtraction, multiplication, and division respectively. It also contains the augend or minuend at the start of an addition or subtraction, and the multiplicand or dividend before multiplication or division.

2.2.6.2 The Q Register contains 37 bits. It holds the low order bits of a double-length dividend, (i.e., the 36 least significant bits of a 72-bit dividend) at the start of a division. At the end of multiplication and division, it contains the low order bits of a double length product, and the quotient, respectively. It holds the mask during a masking operation. It can be used in connection with the accumulator for shift and cycle operations.

2.2.6.3 The B Register contains 37 bits. It is used to hold one of the operands of an arithmetic operation.

2.2.6.4 The Program Counter (PC) contains the address of the next order to be executed. This keeps account of where you are in the program and can be addressed at any time.

2.2.6.5 The Program Counter Store Register (PCS) is used to facilitate the exit from and return to the main program; i.e., the contents of PC can be placed here and then used later to specify sequence if normal sequence has been interrupted.

2.2.6.6 An Index Register is used to modify a coded address by adding its contents to the specified address.

Notation to be used in examples:

$C(\alpha)$ - means the contents of α when α is some address.

$C(\alpha)_{3-8}$ - means the contents of α, bits 3 through 8, excluding everything else.

-----> - means the flow of information is going from left to right.

Example: C(A) -----> Q - means the contents of the accumulator go to the Q register.

Programs are written with octal addresses

3. MOBIDIC ARITHMETIC AND DATA HANDLING INSTRUCTIONS

3.1 Arithmetic Instructions: Addition & Subtraction

CLA $\gamma\ \alpha$ C (α) -----> Acc.

ADD $\gamma\ \beta\ \alpha$ C (A) + C (α) -----> Acc.

C (B) changes as a function of the magnitudes and signs of the operands*.

SUB $\gamma\ \beta\ \alpha$ C (A) - C (α) -----> Acc.

C (B) changes*

ADM $\gamma\ \beta\ \alpha$ C (A) + $|$ C (α) $|$ -----> Acc.

C (B) changes*

SBM $\gamma\ \beta\ \alpha$ C (A) - $|$ C (α) $|$ -----> Acc.

C (B) changes*

Example: Compute (x + y - z) if they are stored in memory locations 100, 101, 102 respectively.

```
CLA 100
ADD 101
SUB 102
```

CAM $\gamma\ \alpha$ $|$ C (α) $|$ -----> Acc.

CSM $\gamma\ \alpha$ $-|$ C (α) $|$ -----> Acc.

CLS $\gamma\ \alpha$ - C (α) -----> Acc.

Example:

Compute $|C(200)| + C(201) - |C(202)| + C(203) - C(204) + |C(205)|$

*See Preliminary Programming Manual, MOBIDIC, page 57, September 1958.

Assume there is no overflow.

There are always alternate methods for programming a problem, each of which may be correct; some may be faster than others and some more efficient in regard to space.

Solution	1st Alternate Method	2nd Alternate Method
CAM 200	CLS 204	CSM 202
ADD 201	ADM 200	ADM 200
SBM 202	ADD 201	ADD 201
ADD 203	SBM 202	ADD 203
SUB 204	ADD 203	SUB 204
ADM 205	ADM 205	ADM 205

Calling for data from a non-existent register address is equivalent to placing all zeros in the memory location and/or register address specified by the instruction.

Example: CLA 70000 clears the accumulator.

3.2 Store and Load Orders

STR $\gamma\ \alpha$ C(A) -----> α

LOD $\gamma\ \beta\ \alpha$ C(α) -----> β

where α is an addressable register or a memory location, and β signifies an addressable register. The machine interprets only the low order 5 bits of the β , or register address. For example, if β were 0512, it would be interpreted as 0012, or the address of the B register. The high order $(7)_8$ is omitted from the β portion, but is assumed by the machine.

Example: Put the difference of C(50) and C(51) in Index Register 1, C(52) in Index Register 2, and the Q Register in Index Register 3. Clear Index Register 4.

LOD 70000,0,0004	Zero -----> IR4
CLA 50	C(50) -----> Acc
SUB 51	C(Acc) - C(51) -----> Acc
STR 70001	Store in IR 1
LOD 52,0,0002	C(52) -----> IR 2
LOD 70011,0,0003	C(Q) -----> IR 3

3.3 Arithmetic Instructions: Multiplication, Division

MLY $\gamma\alpha$	C(α) · C(A) --> A,Q	A - unrounded high order Q - low order	36 bits 36 bits
MLR $\gamma\alpha$	C(α) · C(A) --> A,Q	A - rounded high order Q - low order	36 bits 36 bits
*DVD $\gamma\beta\alpha$	C(A) / C(α) --> Q	Remainder in Acc; no roundoff	
*DVL $\gamma\beta\alpha$	[C(A), C(Q)] / C(α) --> Q	Remainder in Acc; no roundoff	

Example: Compute $\dfrac{(x_1)\ (x_2)\ (x_3)}{(x_4)\ (x_5)}$

where x_i is in 500 + i, $x_4 > x_1$ and $x_5 > x_2$, and all values are fractions.

Put answer in 200.

	Comments
CLA 504	X_4 -----> Acc
MLR 505	$X_4 \cdot X_5$ -----> Acc

*If A is greater than the contents of α, division does not take place.

STR	521	Temporary storage for denominator
CLA	501	X_1 -----> Acc
MLR	502	$X_1 \cdot X_2$ -----> Acc
MLR	503	$X_1 \cdot X_2 \cdot X_3$ -----> Acc
DVD	521	Quotient -----> Q
LOD	70011,0,0010	C(Q) -----> Acc
STR	200	

Example: Compute $\frac{2x^2 - 3y}{2z}$ where x is in 500, y is in 501, and z is in 502. Put result in 503.

		Comments
CLA	502	Z -----> Acc
ADD	502	Z + Z = 2Z -----> Acc
STR	505	Temporary storage for denominator
CLA	500	x -----> Acc
MLR	500	$x \cdot x = x^2$ -----> Acc
ADD	70010	$x^2 + x^2 = 2x^2$ -----> Acc
SUB	501	$2x^2 - y$ -----> Acc
SUB	501	$2x^2 - 2y$ -----> Acc
SUB	501	$2x^2 - 3y$ -----> Acc
DVD	505	Acc ÷ denominator -----> Q
CLA	70011	Get Quotient in Acc.
STR	503	Store in memory

3.4 Arithmetic Operations: Shifts

SHR $\gamma\alpha$ C(A) shifted α (mod 128) places to the right.

A_{sn} not shifted.

SRL $\gamma\alpha$ C(A), C(Q) shifted α (mod 128) places to the right.

A_1 -----> Q_{36}

A_{sn}, Q_{sn} are <u>not</u> shifted.

SHL $\gamma\beta\alpha$ C(A) shifted α (mod 128) places to the left.

A_{sn} not shifted.

Overflow possible.

SLL $\gamma\beta\alpha$ C(A), C(Q) shifted α (mod 128) places to the left.

A_{sn}, Q_{sn} not shifted.

Overflow possible.

Note: Shifts left and right are equivalent to multiplication and division respectively by powers of 2.

Mod 128 - The machine interprets only the low order 7 bits of α for a shift (or cycle) order. The reason for this is that the number of shifts is put in a buffer (the "T" counter) which is 7 bits in length and which can contain the maximum value of $2^7 - 1 = 127$.

Timing of Shifts - The computer will shift right up to 14 places or left up to 9 places in 16 μsec. If for a right shift $\alpha > 14$, or a left shift $\alpha > 9$, then additional time is required.

Example: SHR 36 If $\alpha > 14$, then timing is

$[2 + \alpha - \alpha \pmod 2]\mu$ sec.

α in this case is greater than 14.

$\therefore$ 2 + 36 - 0 (since it is evenly divisible by 2)

$= 2 + 36 = 38\mu$ sec.

Compute: $\dfrac{C(400) - 2C(401) + |C(402)| \cdot C(403)}{8C(404)}$

Conditions: No constants required.

Put answer in location 500 and in the B Register.

Assume there is no overflow.

CAM	402	C(402) ⟹ Acc
MLR	403 (or MLY 403)	C(A) * C(403) ⟹ Acc
ADD	400	C(A) + C(400) ⟹ Acc
SUB	401	C(A) - C(401) ⟹ Acc
SUB	401	C(A) - C(401) ⟹ Acc
DVD	404 (or DVL -)	C(A) ÷ C(404) ⟹ Q
CLA	70011	C(Q) ⟹ Acc
SHR	3	C(A) ÷ 8 ⟹ Acc
STR	70012	Put in B Register
STR	500	Put C(A) in 500

Example: Compute $\dfrac{2x^2 - 3y}{2z}$

This previously used example has been modified to show one use of the shifting operations.

		Comments
CLA	500	$x \longrightarrow$ Acc
MLR	500	$x \cdot x = x^2 \longrightarrow$ Acc
SHL	1	$2x^2 \longrightarrow$ Acc
STR	520	Temporary storage
CLS	501	$-y \longrightarrow$ Acc
SHL	1	$-2y \longrightarrow$ Acc
SUB	501	$-3y \longrightarrow$ Acc
ADD	520	$2x^2 - 3y \longrightarrow$ Acc
SHR	1	$(2x^2 - 3y)/2 \longrightarrow$ Acc
DVD	502	$\frac{2x^2 - 3y}{2z} \longrightarrow$ Q
CLA	70011	Get Quotient in Acc
STR	503	Store in memory

*n shifts left is equivalent to multiplying by 2^n; n shifts right is equivalent to dividing by 2^n.

3.5 Data Handling Instructions: Move and Replace Address

MOV $\gamma\ \beta\ \alpha$ $\quad C(\alpha) \longrightarrow \gamma\beta \quad C(\alpha) \longrightarrow$ B Register

$\alpha \longrightarrow$ Q Register

The move instruction cannot be indexed.

RPA $\gamma\ \alpha$ $\quad C(A)_{1-15} \longrightarrow \alpha_{1-15}$

$C(A) \longrightarrow$ B

The remainder of the C (α) are unchanged.

Example:

Add C(100) $_{1-15}$ to itself and put into location 101, ignoring the overflow into the 16th bit.

CLA	100	C(100) ----> Acc
SLL	1	C(100) * 2 ----> Acc
RPA	101	Acc_{1-15} ----> 101_{1-15}

4. OVERFLOW AND SCALING

4.1 Overflow

Overflow can be caused by any of the following instructions:

ADD	SUB	DVD	SHL
ADM	SMB	DVL	SLL

Overflow is controlled by the 3 low order bits of the β portion, β_{16}, β_{17}, β_{18}. In the chart below, clear (reset) = 0; set = 1.

OVERFLOW CONTROL FOR ARITHMETIC AND SHIFT INSTRUCTIONS

Bits			Before Execution of Instruction	If Overflow Occurs
β_{18}	β_{17}	β_{16}		
0	0	0	Clear OA	Set OA and halt
0	0	1	Clear OA	Set OA
0	1	0	Clear OA	Set OA and halt
0	1	1	Clear OA	No action
1	0	0	No Action	Set OA and halt
1	0	1	No Action	Set OA
1	1	0	No Action	Set OA and halt
1	1	1	No Action	No action

4.2 Scaling

MOBIDIC is a fixed-point computer. This means that the radix point is at some fixed place which is usually, but not always, at the left.

4.2.1 Specifications

a) Scaling requires that all intermediate results, not only the answers, must be scaled.

b) Before starting the computation, determine the maximum and minimum sizes of all variables so as to keep track of the ranges of numbers in the machine.

c) Scaling is done both externally and internally - i.e., in writing equations and coding them. There are no hard and fast rules. These listed below are merely a guide. Juggling is often required.

d) Every time 2 numbers are added, the sum must be < 1 (absolute value); otherwise overflow will occur. If in division the denominator is less than the numerator, overflow will occur.

4.2.2 Rules

a) Numbers to be added or subtracted must have their binary points aligned.

b) To determine the position of the binary point in the result of a multiplication:
Add the number of binary digits to the right of the binary point in the first factor to the number of binary digits to the right of the binary point in the second factor. The sum is the number of bits appearing to the right of the binary point in the product. Remember that when 2 numbers with leading zeros are multiplied together, the total number of leading zeros appears in the result.

c) To determine the position of the binary point in the result of a division:

Subtract the number of bits to the right of the binary point of the divisor from the number of binary bits to the right of the binary point of the dividend. The difference is the number of bits appearing to the right of the binary point in the quotient.

Compute:

$y = \frac{1}{2}(a+b)$

a = 0.9 in location 100

b = 0.6 in location 101

y is in location 102

Code this so as to avoid overflow:

CLA 100	0.9 -----> Acc
SHR 1	0.9 x $\frac{1}{2}$ -----> Acc
STR 500	$\frac{0.9}{2}$ in 500 -----> Acc
CLA 101	0.6 -----> Acc
SHR 1	$\frac{0.6}{2}$ -----> Acc
ADD 500	$\frac{0.9}{2} + \frac{0.6}{2}$ -----> Acc
SHL 1	Adjust answer
STR 102	Put in 102

Evaluate:

$$f(x) = 3.2x^2 - 294x + 0.001$$

for $0 < x < 100$ in steps of 1.

$$f(x) = 1000(0.0032x^2 - 0.294x + .000001)$$

Let $y = \frac{x}{100}$; then $x = 100y$

$$f(x) = 1000\left[0.0032y^2(100)^2 - 0.294(100)y + 0.000001\right]$$

$$= 10^{+7}(0.0032y^2 - 0.00294y + 0.0000\ 000\ 001)$$

This exemplifies external scaling.

Example:

0.625 0.625 in location 101
0.57 0.57 in location 102

CLA 101	0.625 ----->	Acc
SHR 1	0.0625 ----->	Acc to insure division
DVD 102	C(A) ÷ 0.57 ----->	Q
CIA 70000	Clear the accumulator	
SLL 1	Makes up for the shift right above, puts integer in A while fraction stays in Q.	

4.3 Normalize Order

NRM γ α shifts the contents of the accumulator left until a one appears in bit 36. The number of necessary shifts is stored in the low order bits of the location specified by α of this instruction. The normalized number stays in the accumulator.

Example:

Compute x/y, where both values are positive and x > y.

			x is in location 100; Location 1 contains 1 y is in location 101
500	CLA	101	y -----> Acc
501	NRM	102	Normalized count -----> 102
502	STR	103	Normalized value -----> 103
503	CLA	100	x -----> Acc
504	NRM	104	Normalized count -----> 104
505	SHR	1	To insure division, shifting numerator to put 0 in.
506	DVL	103	x/y -----> Q
507	MOV	70011, 00105	C(Q) -----> 105
510	CLA	102	Normalized count of denominator -----> Acc
511	SUB	104	Acc - normalized count of numerator -----> Acc
512	SUB	1	To make up for earlier SHR 1.
513	RPA	515	Put in address of shift instruction
514	CLA	105	x/y -----> Acc
515	SHR	**	Puts binary point in right place. This would be a SHL if the normalized count of the numerator was greater than the normalized count of the denominator.
516	STR	106	Acc -----> 106

The absolute difference between the numerator and denominator -1 will give the exact amount of places which one has to shift in order to put the binary point in the right position.

Homework:

Problem #1. Solve for x and y

$$1286x - 1195y = 421$$

$$1395x + 1003y = 721$$

where

C(400) = 1286

C(401) = 1195

C(402) = 1395

C(403) = 1003 Integers (Decimal numbers)

C(404) = 421

C(405) = 721

Solving the equations:

$$y = \frac{1286(721) - 421(1395)}{1286(1003) + 1195(1395)}$$

$$x = \frac{721 - 1003y}{1395}$$

100	CLA	400	1286 -----> Acc
101	MLY	403	1286(1003) -----> Q
102	MOV	70011, 00410	C(Q) -----> 410
103	CLA	401	1195 -----> Acc
104	MLY	402	1195(1395) -----> Q
105	CLA	70011	C(Q) -----> Acc
106	ADD	410	1286(1003) + 1195(1395) -----> Acc
107	STR	410	Store denominator in 410
110	CLA	400	1286 -----> Acc
111	MLY	405	1286(721) -----> Q
112	MOV	70011, 00411	C(Q) -----> 411

113	CLS	404	-421 -----> Acc
114	MLY	402	(-421)(1395) -----> Q
115	CLA	70011	C(Q) -----> Acc
116	ADD	411	1286(721) - 421(1395) -----> Acc
117	DVD	410	Divide by denominator
120	MOV	70011, 00512	y -----> 512
121	CLS	403	-1003 -----> Acc
122	MLY	512	(-1003)(y) -----> Q
123	CLA	70011	C(Q) -----> Acc
124	ADD	405	C(Acc) + 721 -----> Acc
125	DVD	402	721 - 1003y/1395 --> Q
126	MOV	70011, 00513	x -----> 513

Problem #2: Given

A in location 200 - binary point between 9 and 10 with 20 leading zeros.

B in location 201 - binary point between 14 and 15 with 18 leading zeros.

C in location 202 - binary point between 6 and 7 with 23 leading zeros.

D in location 203 - binary point between 3 and 4 with 19 leading zeros.

Compute: $x = \frac{(A)(B)}{(C)(D)}$

Put Integer part of answer in location 210 and put Fraction part of answer in location 211.

Multiplying (A)(B) the answer appears in Q with 2 leading zeros. Move to Acc (38-36= 2)

Multiplying (C)(D) the answer appears in Q with 8 leading zeros. Move to Acc (44-36= 8)

Numerator - Denominator -1 (which, as previously explained represents a numerator shift to avoid overflow) = the number of shifts necessary at end of computation to place the radix point correctly. (2-8-1= -7). After division, a SRL 14 will put the binary point between the Q and Acc. This is the external calculation; the first solution is based on this.

100	CLA 202	C -----> Acc
101	MLY 203	(C)(D) -----> Acc, Q; all 0's in Acc
102	CLA 70011	C(Q) -----> Acc
103	NRM 204	Normalize denominator
104	STR 205	Store normalized value
105	CLA 200	A -----> Acc
106	MLY 201	(A)(B) -----> Acc, Q; all 0's in Acc
107	CLA 70011	C(Q) -----> Acc
110	NRM 206	Normalize numerator
111	SHR 1	Assure division without overflow
112	DVD 205	$\frac{(A)(B)}{(C)(D)}$ - all 0's in Acc
113	CLA 70011	C(Q) -----> Acc
114	SRL 7	[SRL 14 - SHL 7]
115	STR 210	Store integer
116	MOV 70011, 00211	Store fraction

Using internal calculation, put the 14 (= number of shifts needed to adjust binary point to the proper place) in location 500 and substitute in the previous program the following:

114	STR 477	Temporary storage for answer
115	CLA 206	Numerator count

```
116  SUB  204          Denominator count
117  SUB  1            Make up for the SHR 1 previously.
                       (Assume that the contents of location
                       1 is a 1)
120  RPA  122          Put the count in the address part
                       of shift instruction.
121  CIA  477          Put answer back in Acc
122 *SRL  ---          Shift stored number of times
123  STR  210          Store integer
124  MOV  70011, 00211 Store fraction
```

*After subtracting the denominator from the numerator -1, the answer may be positive or negative, shifting left or right respectively. However, in internal calculation, since the sign is unknown, a conditional transfer should be used. Substitute the following in the previous program:

```
120  TRP  125
121  RPA  123          If the sign is negative, shift right.
122  CIA  477
123  SRL  ---
124  TRU  130
125  RPA  127          If the sign is positive, shift left.
126  CIA  477
127  SLL  ---
130  STR  210
131  MOV  70011, 00211
```

5. LOOPS

Compute $X^2 + 0.1$ for $0.2 \leq X < 0.5$ in steps of .01 and put on tape.

Location 77 contains 0.1

" 100 contains 0.2

" 101 contains 0.01

" 102 contains 0.2

" 103 contain 0.5

Method 1:

200	CLA	100	0.2 → Acc
201	MPY	70010	(.2)(.2) → Acc
202	ADD	77	(.2)(.2) + .1 → Acc
203	Put on Tape		
204	CLA	100	0.2 → Acc
205	ADD	101	0.2 + 0.01 → Acc
206	STR	100	0.21 → loc 100
207	MPY	100	(.21)(.21) → Acc
210	ADD	77	(.21)(.21) + .1 → Acc
211	Put on Tape		
212	CLA	100	0.21 → Acc
213	ADD	101	0.21 + 0.01 → Acc
214	STR	100	0.22 → loc 100
215	MPY	100	(.22)(.22) → Acc
216	ADD	77	(.22)(.22) + .1 → Acc

217 Put on Tape

220 Continue until .5 value for X is reached.

This method is long and inefficient, especially when there are many values. The efficiency of this program can be improved by using control transfers.

5.1 Control Transfers

TRU $\gamma\ \beta\ \alpha$ — $\alpha \rightarrow$ PC; γ is used for indexing; β is used for trapping mode control.

TRN $\gamma\ \alpha$ — If $A_{sn} = 1$, $\alpha \rightarrow$ PC; If $A_{sn} = 0$, continue in sequence.

TRP $\gamma\ \alpha$ — If $A_{sn} = 0$, $\alpha \rightarrow$ PC; If $A_{sn} = 1$, continue in sequence.

TRZ $\gamma\ \alpha$ — If $C(A)_{1-36} = 0$, $\alpha \rightarrow$ PC; If $C(A)_{1-36}$ are not $= 0$, continue in sequence.

Method 2: Using a control transfer to create a loop.

200	CLA	100	0.2 → Acc; 0.2 = X
201	MPY	70010	(.2)(.2) → Acc
202	ADD	77	(.2)(.2) + .1 → Acc.
203	Put on tape		
204	CLA	100	X → Acc
205	ADD	101	X + 0.01 → Acc
206	STR	100	X is increased by 0.01 with each additional loop.

```
207    MPY    100        (x)(x)  →  Acc
210    ADD    77         (x)(x) + 0.1  → Acc
211    Put on Tape
212    TRU    204        Go back and repeat
```

Method 2 is a "closed loop"; that is, it will continue forever because there is no test for completion.

Method 3: Using a completion test to end the loop.

```
200    CLA    100
201    MPY    70010
202    ADD    77
203    Put on tape
204    CLA    100
205    ADD    101
206    STR    100
207    MPY    100
210    ADD    77
211    Put on tape
212    CLA    100        Completion test.  When x reaches
213    SUB    103        the value of 0.5, the loop is
                          finished.
214    TRN    204
215    HLT
```

Method 3 takes longer than method 1, but takes less space.

Method 3 also should be initialized to facilitate re-entry.

Method 4: Adding initialization to method 3.

177 MOV 102, 100

200 Continue with method 3.

Example: Add the contents of positions 1000_8 - 4000_8 ignoring overflow.

```
        170   storage for partial sums
        171   ADD   1000      Initializing constant   }
        172   + 1                                     }  Constants
        173   ADD   4001_8    Terminating constant    }

START — 200   MOV   70000,170 0 → 170                 }  Initialization
        201   MOV   171,203   (ADD 1000) → 203        }
        202   CLA   170                               }
        203   (ADD  1000)                             }  Computation
        204   STR   170       C(1000) → 170           }
        205   CLA   203       ADD 1000 → Acc          }
        206   ADD   172       Change to ADD 1001      }  Address
        207   RPA   203       and put in 203          }  modification
        210   SUB   173       ADD 1001 - ADD 4001     }  Test for
        211   TRN   202                               }  Completion
        212   HLT
```

5.2 Add Beta and Subtract Beta Orders

Locations 205, 206 and 207 exemplify the technique called address modification. Two instructions which facilitate address modification are Add Beta and Subtract Beta.

ADB $\gamma\ \beta\ \alpha$ $\quad C(A) \rightarrow Q$

$\beta + C(\alpha)_{1-12} \rightarrow A, \alpha, I^{\gamma}$

B register depends on $C(A), C(\alpha)$

SBB $\gamma\ \beta\ \alpha$ $\quad C(A) \rightarrow Q$

$C(\alpha)_{1-12} - \beta \rightarrow A, \alpha, I^{\gamma}$

B register depends on $C(A), C(\alpha)$

In reference to the previous problem, three instructions can be replaced using ADB.

205	CLA	203	ADB 203,0,0001
206	ADD	172	
207	RPA	203	

Example: There are 100 numbers stored in loc. 500 to 599. Store the number of zeros in loc. 170, and the difference between the number of plus's and minus's in location 160.

			Comments
600	CLA	70000	Clear Acc
601	STR	160	Initialize storage location
602	STR	170	" " "
603	CLA	500	First number

```
604   TRZ   610
605   TRN   612
606   ADB   160,0,1   If +, add 1 to loc 160
607   TRU   613
610   ADB   170,0,1   If 0, add 1 to loc 160
611   TRU   613
612   SBB   160,0,1   If -, sub. 1 from loc 160
613   ADB   603,0,1   Puts 501, 502, --599, in 603
614   SUB   617       Test for completion
615   TRN   603       Transfer if not finished
616   HLT             Stop
617   CLA   600       Constant for completion test.
```

5.3 Sense Orders

SEN $\gamma\beta\alpha$ If the flip-flop specified by β is equal to 1, $\alpha \rightarrow$ PC. Otherwise, the program continues in sequence.

Example: x is in location 50

y is in location 51

If x + y = 1 go to location 400 } These cases

" x + y > 1 go to location 500 } will cause overflow

" x + y < 1 go to location 600

```
60)   CLA  50   x → Acc
61)   ADD  51   C(A) + y → Acc
```

Overflow can occur at this point. How can the programmer control this overflow? See chart under overflow section in notes. One alternative would be:

60) CLA 50

61) ADD 51,0,0001

The 1 in the beta part of this instruction clears the overflow alarm before the execution of the instruction, and sets the overflow alarm if overflow occurs.

62) SEN 64,0,0100 Senses OA, if not set, continues

63) TRU 600

64) TRN 600 Overflow was negative

65) TRZ 400 $x + y = 1.0$ $1\boxed{0 \cdots 0}$ ∴ Acc = 0

66) TRU 500 A TRP would do the same thing.

If we interchanged locations 64 and 65, the program would not be correct, because the TRZ instruction does not test the sign of the Accumulator.

SEN merely tests the flip-flop, it does not change it. There are two other instructions which alter the flip-flops:

SNS $\gamma\beta\alpha$ If the flip-flop specified by β is equal to 0_γ set it to 1, and $\alpha \longrightarrow PC_\gamma$ Otherwise, continue in sequence.

SNR $\gamma\ \beta\ \alpha$ If the flip-flop specified by β is equal to 1, set it to 0, and $\alpha \longrightarrow$ PC. Otherwise, continue in sequence.

Note: On SNS and SNR, if the flip-flop is changed, then $\alpha \longrightarrow$ PC. If the flip-flop is not changed, then PC + 1 $\longrightarrow$ PC.

Example: 100 SNS 101,0,110

101

SNS used in this manner insures that the first sense flip-flop is set.

Example: If $x + y < 1$, compute x^2 and store in location 500 unless $x < 0$, in which case compute y^2 and store in location 501. x is in location 100 and y in location 101.

```
200  CLA   100        x ⟶ Acc
201  TRN   210        Transfer if x < 0
202  ADD   101,0,1    x + y and check for
                      overflow
203  SNR   207,0,0100 If overflow, no computation;
                      go to halt
204  CLA   100        No overflow here, ∴ x ⟶ Acc
```

205	MLR	100	$x \cdot x = x^2$
206	STR	500	$x^2 \longrightarrow 500$
207	HLT		
210	CLA	101	If negative, want to compute
			$y^2 \therefore y \longrightarrow$ Acc
211	MLR	101	$y \cdot y = y^2 \longrightarrow$ Acc
212	STR	501	$y^2 \longrightarrow 501$

Example: Compute C(200) + C(201) - $|C(202)|$ + C(203) and put the result in location 300. Check for overflow wherever possible. If there is an overflow, set the OA and go to an error routine which corrects this overflow and returns into the program.

500	CLA	200	C(200) → Acc
501	ADD	201,0,1	C(200) + C(201) → Acc
502	SNS	504,0,0100	Overflow?
503	TRL	514	Yes
504	SBM	202,0,1	No. - C(200) + C(201) - $\|C(202)\|$
505	SNS	507,0,0100	Overflow?
506	TRL	514	Yes
507	ADD	203,0,1	No. - C(200) + C(201) - $\|C(202)\|$ + C(203)
510	SNS	512,0,0100	Overflow?
511	TRL	514	Yes

```
512   STR   300      No. Store
513   HLT
514   ⇌        ←     Here is a routine which
                     corrects the overflow and
                     returns (TRS) to the
                     proper place in the
                     program.
```

5.4 Compare Order

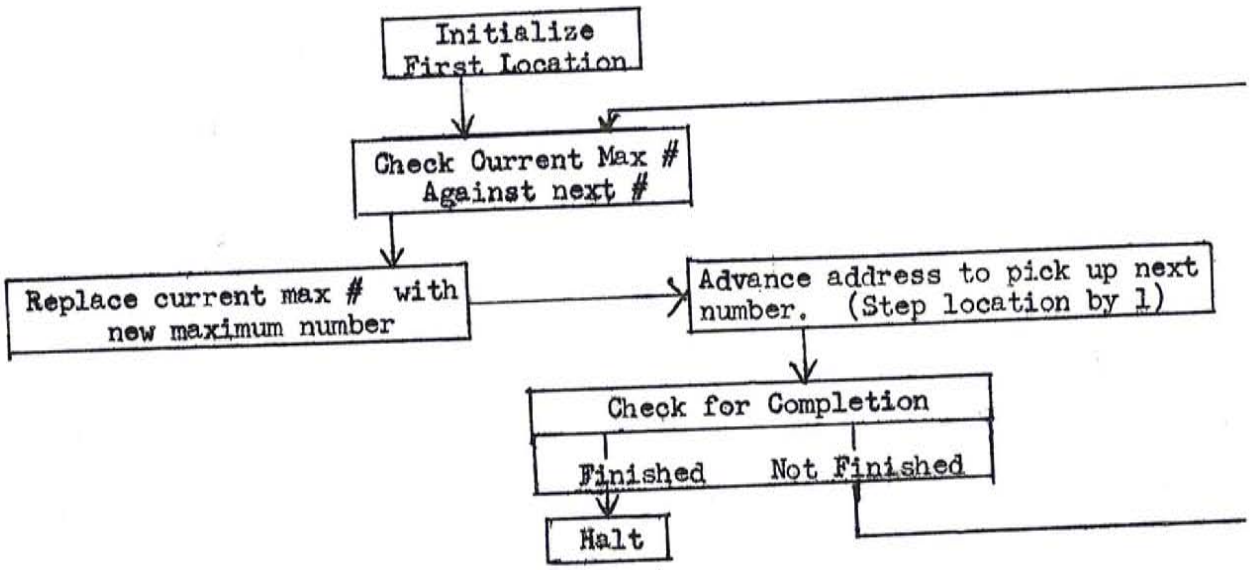
TRC $\gamma\alpha$ $C(A) < C(\alpha) \Longrightarrow PC + 1 \longrightarrow PC$ (continue in sequence)

$C(A) > C(\alpha) \Longrightarrow PC + 2 \longrightarrow PC$

$C(A) = C(\alpha) \Longrightarrow PC + 3 \longrightarrow PC$

Example: Find the largest number in positions 100-200.

Before coding, flow chart the problem.



Coding:

```
576  MOV  614,600   }  Initialization
577  MOV  615,601   }
600  CLA  100       First number in Acc
601  TRC  101       Compare with 2nd number
602  TRU  610       C(A) < C(∝)
603  TRU  604       C(A) > C(∝)
604  ADB  601,0,1   C(A)= C(∝) Step counter for
                         another comparison
605  SUB  616       Check for completion
606  TRN  600       Transfer if negative
607  HLT            Halt if completed
610  CLA  70012     If greater than, put
                    address of larger number
                    in Acc.
611  RPA  600       Put larger number in 600
612  TRU  604       Step up comparison
614  CLA  100       Initializing constant
615  TRC  101       Initializing constant
616  TRC  201       Terminating constant
```

Homework -

Consider the positive numbers which are in positions 100_8 - 200_8 inclusive. There is a number in location 77_8.

Move all the numbers which are larger than the $C(77_8)$ to consecutive locations starting in 300_8. Record in locations 201_8, 202_8, 203_8 respectively the number of values which are greater than, less than and equal to $C(777)_8$.

573	MOV	70000,201	
574	MOV	70000,202	Initialization (loc. 573 through 577)
575	MOV	70000,203	
576	MOV	701,600	
577	MOV	702,610	
600	CLA	[100]	100 in Acc
601	TRC	77	Compare with C(77)
602	TRU	606	C(Acc) < C(∝)
603	TRU	610	C(Acc) > C(∝)
604	ADB	203,0,1	C(Acc) = C(∝); add one to = counter
605	TRU	613	
606	ADB	202,0,1	Add one to < counter
607	TRU	613	
610	STR	[300]	Store > in 300 ff
611	ADB	610,0,1	Step up by one, storage location for >
612	ADB	201,0,1	Add one to > counter
613	ADB	600,0,1	Step up next number for comparison.

614	SUB	700	Terminating constant
615	TRN	600	Test for completion
616	HLT		
700	CLA	201	Terminating constant
701	CLA	100	Initializing constant
702	STR	300	Initializing constant

5.5 Indexing: LDX and TRX Instructions

Indexing is a way of accomplishing address modification. By the use of an index register, the coded address of an instruction is modified for operating purposes, but the instruction is not changed in memory.

Example: CLA 500,1

If index register 1 contains a 5, then the effective address would be CLA 505 = (500 + 5). The instruction itself is not disturbed and, therefore, need not be initialized.

The "Effective Address" of an indexable order is equal to the actual (stated) address + the contents of the specified index register. The orders which affect index registers are:

LDX $\gamma\ \beta\ \alpha$ $\beta \longrightarrow I^{\gamma}$

$\alpha \longrightarrow I^{\gamma+1}$

Example: $\alpha\ \gamma\ \beta$

LDX 2,3,5 $\qquad$ $5 \longrightarrow I^3$

$2 \longrightarrow I^4$

The gamma portion is interpreted modulo 4. If gamma = 4, gamma + 1 = 1.

When a non-existent index register is addressed by LDX (e.g. LDX 2, 0, 5), the instruction is considered a no-op.

Note that the α of the LOD instruction is an address, but that both the α and β of a LDX instruction are the actual numbers to be loaded.

TRX $\gamma\ \beta\ \alpha$ If $C(I^{\gamma+1})=0$, continue in sequence.

If $C(I^{\gamma+1})$ are not = 0, then $C(I^{\gamma+1}) - 1 \longrightarrow I^{\gamma+1}$

If now, $C(I^{\gamma+1}) = 0$, continue in sequence

If $C(I^{\gamma+1})$ are not = 0, $C(I^{\gamma}) + \beta \longrightarrow I^{\gamma}$ and $\alpha \longrightarrow$ PC. (next instruction is taken from α).

Note: $\alpha \longrightarrow$ B Register (even though a non-existent index register may be addressed).

Examples: Add the contents of 1000_{10} - 4000_{10} ignoring overflow.

```
500)   CLA      70000
  1)   LOD      505,0,0001    (3001)g --> IR1
  2)   ADD      777,1,7
  3)   TRX      502,4,0       Decrease IR1 by 1
                               Don't change IR2
  4)   HLT
  5)   +3001g
```

Move the block of 50 numbers in 101 - 150 to locations 126 - 175 using 2 index registers.

```
500)   LDX      50,1,1        50 --> IR2,1 -->IR1
       CLA      100,2
       STR      125,2
       TRX      501,1,1
       HLT
```

Find out how many positive numbers are in the set of numbers from location 105 to location 142.

```
1000)  MOV      70000,1050    Clear counter
  1)   LOD      1010,0,0001   38 --> IR1
  2)   CLA      104,1
  3)   TRP      1006
  4)   TRX      1002,4,0
  5)   HLT
  6)   ADB      1050,0,1
  7)   TRU      1004
1010)  +38(10)
```

The answer appears in the counter, location 1050.

There are six instructions which, can modify one or more index registers:

TRL, ADB, SBB - change I^{γ}

LDX, TRX - change I^{γ} and $I^{\gamma}+1$

RPT changes I^3 and I^4.

CLASS PROBLEM:

Given: x in position 100 and an integer n in position 101. Compute x^n, using an index register, and leave the answer in the accumulator. Assume x is a fraction.

```
200   SBB   101,2,1   puts n-1 in I2
201   CLA   100
202   MPY   100
203   TRX   202,1,0
204   HLT
```

This program illustrates how precise one must be in loading the correct number into the index register.

6. MACHINE LOGIC

6.1 Boolean Algebra: Or, And, and Complementing Operations.

6.1.1 The rules for Logical Addition otherwise known as the "or" function are:

$1 + 1 = 1$

$1 + 0 = 1$

$0 + 1 = 1$

$0 + 0 = 0$

where the plus sign indicates "or".

6.1.2 The rules for Logical Multiplication otherwise known as "and" function are:

$(1)(1) = 1$

$(1)(0) = 0$

$(0)(0) = 0$

$(0)(1) = 0$

6.1.3 The rule for Logical Negation or the complement function is:

$$\text{if } Acc = 0,\ Acc' = 1.\ (Acc')' = Acc$$

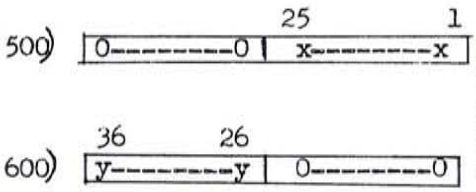
6.2 Logical Instructions

6.2.1 LGA $\gamma\alpha$ C(Acc) are "orred" to C(α) and the result appears in the accumulator. This includes the sign bit.

6.2.2 LGM $\gamma\alpha$ C(Acc) are "anded" to C(α) and the result appears in the accumulator. This includes the sign bit.

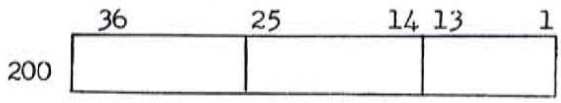
6.2.3 LGN α, $C'(\alpha)$ are complemented and the result appears in the accumulator.

Examples:



Put bits 1-25 of location 500 and bits 26-36 of location 600 into the same word and store in location 700.

```
100  CLA  500
101  LGA  600
102  STR  700
```



Unpack bits 14-25 of location 200 and store it in location 203

```
400  CLA  200
401  LGM  404
402  STR  203
403  HLT
404  OCT  +000177760000
```

	36 26	25 15	14 1
200	x------x	garbage	x------x
201	0------0	y-----y	0------0

Put C(201)$_{15-25}$ into 200$_{15-25}$ and do not disturb bit positions 1-14, 26-36 of loc. 200.

Comments

				Acc
500	CLA	200	x-----x garbage x-----x	Result of LGM
501	LGM	510	x-----x 0-----0 x-----x	(AND)
502	LGA	201	x-----x y-----y x-----x	Result of LGA (OR)
503	STR	200		
504	HLT			
510	777600037777		1-----1 0-----0 1-----1	

7. Editing Orders -

Cycling & Masking

CYS $\gamma\alpha$ — The accumulator cycles the contents of the accumulator left the number of places specified by α (mod 128). The sign is not included.

CYL $\gamma\alpha$ — The accumulator and Q register cycle to the left the number of places specified by α (mod 128). Both A_{sn} and Q_{sn} are included.

MSK $\gamma\alpha$ — For every bit in Q that is a 1, the contents of the corresponding bit of α are replaced by the same corresponding bit of Acc; i.e.,

If $Q_{12} = 1$, and $A_{12} = 0$, $0 \longrightarrow \alpha 12$

If $Q_{12} = 0$, $\alpha 12$ is undisturbed regardless of the state of A_{12}.

Logical function is $AQ + Q'\alpha$.

Example:

Before mask instruction:

C(Acc) = 001 010 011 100

C(Q) = 000 011 000 010

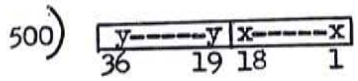
C(α) = 100 011 010 001

after the mask instruction:

C(α) = 100 010 010 001

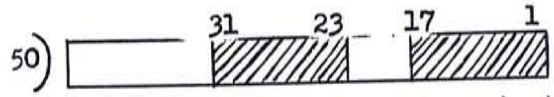
Problems:

In location 500 there is a number xy. Bits 1-18 = x, 19-36 = y. Put x in location 502 and y in location 503, both in bits 1-18. Assume 502 and 503 contain zeros initially.




```
100  CLA  500          C(500) ⟶ Acc
101  LOD  106,0,0011   C(106) ⟶ Q
102  MSK  502          Puts x's in 502
103  SHR   18          Puts y's in position for
                       masking
104  STR  503          Puts y's in 503
105  HLT
106  +000000777777
```

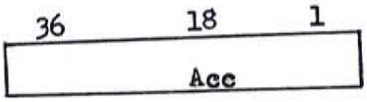
At location 50, there are four pieces of information:



Multiply the shaded area by 2 and store in the same relative positions in location 55. Assume there is no overflow into bit positions 18, 23, and 32.

```
100  LOD  106,0,0011
101  CLA  50
```

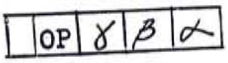
```
103   ADD    50
104   MSK    55
105   HLT
106   017760377777
```



Reverse the upper and lower half of the accumulator, store in location 500 and then put bit 18 of the accumulator into Q_{36}, without changing the sign of the Q.

```
CYS   18
STR   500
CYS   1      The CYL will not work because it
SRL   1       would change Qsn
```

Location

500 

Store α of location 500 in α of 501

Store γ of location 500 in α of 503

Store β and OP of location 500 in the same positions of 502.

```
466   CLA   500          Puts C(500) in Acc
467   RPA   501          Puts α in α of 501
470   LOD   477,0,0011  Mask ⟶ Q
471   MSK   502          Puts β and OP in 502
472   SHR    28          Puts γ in α position
473   LOD   0011,0,476  Mask ⟶ Q
474   MSK   503          Puts γ in α of 503
475   HLT
476   +  0-----07
477   +  77077770-----0
```

8. REPEAT ORDER AND SPECIAL COMBINATIONS

8.1 Repeat Order

RPT $\gamma\ \beta\ \alpha$

The instruction following the repeat instruction is executed $\alpha + C(I\gamma) + 1$ times where α and γ are those of the repeat instruction. The effective address of the i^{th} execution of the order following the repeat instruction is $\alpha + C(I^{\gamma}) + (i-1)\beta$.

Examples:

```
RPT  5,2,1
ADD  100
```

I1	I2	I3	I4
5	2	7	1

```
ADD  100
ADD  101
ADD  102
ADD  103
ADD  104
ADD  105
ADD  106
ADD  107
```

First ADD not under repeat control

Next seven ADDs are under repeat control

```
RPT  5,2,1
ADD  100,1
```

I1	I2	I3	I4
5	2	7	1

```
ADD  105
ADD  106
ADD  107
ADD  110
ADD  111
ADD  112
ADD  113
ADD  114
```

Effective address for ADD 100,1

Next seven ADDs are under repeat control

Example:

$C(100)^2 \cdot C(102) \cdot C(104) \cdot C(106)$. Assume all numbers are fractions.

```
CLA  100
RPT  3,0,2
MPR  100
HLT
```

In detail, notice that MPR is executed 3 + 0 + 1 times, and incremented by 2 after each execution.

Homework Problem:

Considers numbers in 1000_8 to 2000_8. Compute -

C(1000)·C(1001).....C(2000) and put the product in location 3000_8

C(1001).C(1002).....C(2000) and put the product in location 3001_8

C(1002).C(1003).....C(2000) and put the product in location 3002_8

.
.
.
.

C(1050)·C(1051).....C(2000) and put the product in location 3050_8

```
200   LDX   51,1,0
201   CLA   1000,1
202   RPT   726,2,1
203   MLY   1001,1
204   STR   3000,1
205   TRX   201,1,1
206   HLT or continue the program
```

8.2 Repeat-Move

RPT $\alpha, \gamma_1, \beta_1$ When the MOV instruction follows a RPT

MOV $\alpha_2, \gamma_2, \beta_2$ instruction, it is executed $\alpha + C(I^{\gamma}) + 1$ times. The α part of the MOV order is increased by β of the repeat instruction, after each execution of that instruction.

The $\gamma\beta$ part of the MOV order is Indexed by I^2 on each execution after the first.

Before each MOV order is executed, its effective α is placed in Q.

First time (not under repeat order) $\alpha_2 \rightarrow \gamma_2\beta_2$

Second time $C(\alpha_2+\beta_1) \rightarrow [\gamma_2\beta_2 + C(I^2)]$

Third time $C(\alpha_2 + 2\beta_1) \rightarrow [\gamma_2\beta_2 + 2C(I^2)]$

For example: Move every other number in positions 100_{10}-200_{10} inclusive to every 4th location starting at 300_{10}.

```
LDX   0,2,4      4 → I²   0 → I³
RPT   50,0,2
MOV   100,300
HLT
```

8.3 Repeat - Compare **

In the comparison,

If C(A) < C(x)

then x → B Register, transfer control to PC + 1

If C(A) = C(x)

then x → B Register, transfer control to PC + 3

If C(A) > C(x) :

** See Chart, Preliminary Programming Manual, MOBIDIC page 33, September, 1958

If R (number of repetitions expressed by $\alpha + I\delta$ of RPT order) = 0, then transfer to PC + 2

If R>0, then R is decreased by 1, x is increased by β of TRC order, a new comparison is begun.

Problem:

We have a block of 100 numbers starting in location 50. Compare these numbers with a number x in location 500. If there are any numbers which are greater than or equal to x, stop the computer; otherwise continue the program.

```
400  CLA  500
401  RPT  1438,0,1
402  TRC   50
403  HLT
404  TRU  406
405  TRU  403
406  continue program
```

1	2	3	4	= I Register
0	0	99	1	→before first compare
0	0	0	1	→after last compare

Another Problem:

Compare Z, a number in location 500, with C(50) to C(200). Tabulate the number of values which are = Z, using RPT - TRC. Stop when finished.

```
577  MOV  70000,703     Initialize location 703
600  CLA  500           Z ——→ Acc.
601  RPT  150,0,1
```

```
602  TRC   50          x ---> B Register
603  TRU   606         Where locations 601 and
                       602 are adjusted.
604  HLT               Exit when comparison is
                       finished.
605  TRU   1105        Where # of values = Z will
                       be tabulated.
606  CLA   70003       Examine number of repeats.
607  TRZ   604         If 0, comparisons are
                       finished; exit.
610  RPA   601         If not 0, decrease the
                       number of comparisons
                       remaining.
611  ADB   70012,0,1   Bring up the next number
                       for comparison.
612  RPA   602
613  TRU   600
1105 ADB   70012,0,1   C(B) + 1 ---> Acc
1106 RPA   602         Brings up next number for
                       comparison
1107 MOV   70000,700   Initialize location 700
1110 RPA   700         700 = temporary storage
                       location.
```


R

1111	MOV	70000,701	Initialize loc. 701
1112	CLA	601	RPTα, 0,1 $\longrightarrow$ Acc
1113	RPA	701	$\alpha \longrightarrow$ C(701)
1114	CLA	701	$\alpha \longrightarrow$ Acc
1115	SUB	700	In which the next position for comparison is located
1116	ADD	702	Where in loc. 702 is a constant of $\pm$50. In calculating the last three instructions, the actual amount of comparisons remaining are calculated.
1117	RPA	601	Replace these remaining calculations back into the RPT order.
1120	ADB	703,0,1	Steps counter by 1
1121	TRU	600	

9. SUBROUTINE LIBRARY

9.1 Subroutine Transfers and their Operation

TRL $\gamma\ \beta\ \alpha$ PC + 1 ----→ PCS

β ----→ Iγ

α ----→ PC

TRS C(PCS) ----→ PC

Example: $x/\sqrt{y}$ -1. Assume x and y are fractions stored in locations 50 and 51 respectively. Use a square route sub-routine which begins at location 100 with its entrance and exit in the accumulator.

```
500  CLA   51
501  TRL   100,1,501      100)Square Root Routine
502  DVD   50
503  CLA   70011
504  SUB   507                TRS (indicates end of sub-
505  STR   52                     routine and return to
506  HLT                          program counter)
507  + 1
```

Problems:

Making use of a sub-routine:

Location	Operation	$\alpha\gamma\beta$	Comments
125	TRL	500,1,125	
		x	x = location of argument
128		y	y = location of result

normal return

One Method:

Location	Operation	α & β	Comments
500	CLA	1,1,0	Same as CLA 126
501	RPA	504	Puts location of x in 504
502	CLA	2,1,0	Same as CLA 127
503	RPA	520	Puts location of y in 520
504	CLA	[loc. x]	
•	•		
•	•		
•	•		
•	•		
•	•		
•	•		
520	STR	[loc. y]	
521	ADB	504,0,2	Prepare for normal return
522	STR	70014	to main program
523	TRS		

Example:

$x = \sqrt{2a - 1}$ Where a is stored in location 200
and 1 is stored in location 202.
Store answer in location 201

450	CLA	200	α ----> Acc
451	ADD	70010	$\alpha + \alpha$ ----> Acc
452	SUB	202	$2\alpha - 1$ ----> Acc
453	TRL	500	Go to square root sub-routine
			available at location 500
456	STR	201	

```
457         HLT
500         (Sub-routine for Square Root)
---
---
542         TRS
```

9.2 Floating Point Operations

A floating point number is represented by the expression

$$N = X \cdot b^y$$

where b is a positive integer > 1 and $|X| < 1$

X is called the mantissa or fractional part

y " " " exponent part

Examples:

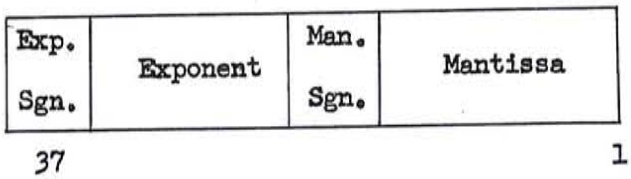
$$.000405 = .405 \times 10^{-3}$$

$$23.49 = .2349 \times 10^{2}$$

$$4.8 = .6 \times 2^{3}$$

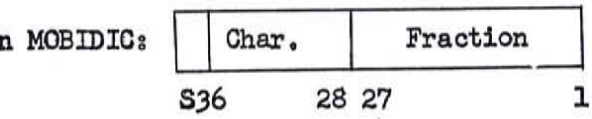
A floating point number can be expressed in a computer in two different ways. The first method uses one word for the exponent and another for the mantissa. This is known as "two word" or "unpacked" floating point. The second method uses one computer word for both the exponent and the mantissa. This is known as "one word" or "packed" floating point. In this second case there is a problem concerning the signs. A computer word normally has only one sign associated with it. When "one word" floating point is used, two signs are needed; one sign for the exponent and the other sign for the fraction.

This necessity of two signs can be handled by allowing the normal sign bit for the exponent sign and another bit for the fraction sign. This is illustrated by



Since the programmer may at times find this method awkward, the concept of a "characteristic" is introduced. The number of bits allowed for the exponent determines the maximum positive and negative values. The characteristic is defined as the exponent plus a constant which is one more than the absolute value of the smallest exponent. For example, if the exponent was between -255 and +255 then the corresponding characteristics are between 1 and 511. Thus, every negative exponent is replaced by a positive characteristic and the sign of the computer word applies only to the mantissa.

MOBIDIC uses an Excess 256_{10} system,



a) The fraction part is contained in bit positions 1 through 27. A floating point number having 1 bit in position 27 is said to be normalized.

b) The sign bit indicates the algebraic sign of the

fraction and, since signed exponents are desirable, the characteristic is formed by adding +256 to the exponent. Thus the range of the characteristic is $0 \leq C \leq 511$ while the range of the exponent is $-256 \leq EXP \leq 255$.

9.3 Double Precision

A fractional number in double precision is represented by

$N = x + y \cdot b^{-n}$, where b is the base and n is the size of the machine word

Example: In a two digit machine the number

$.2594 = .25 + .94 \times 10^{-2}$

$= .25 + .0094$

∴ Two storage locations are needed to represent a double precision number.

Example: Adding two numbers -

2594

3024

	1) 94	26	25 (+ 1) = 26
	24	30	
carries over	(1) 18	56	∴ 56 18

10. TYPICAL EXAM

Assume a set of 800 positive numbers are stored in memory beginning at location 500_{10}. There is a positive number Z stored at location 400_{10}. Count the numbers less than and equal to Z and store the counters in 401 and 402 respectively. Add the numbers greater that Z, keeping track of the overflow. Store the cumulative sum in 403 and the amount of overflow, if any, in 404.

74	MOV	70000,401	initialization of less than $\leq$ counter
75	MOV	" 402	initialization of equal to $\leq$ counter
76	MOV	" 403	initialization of cumulative sum storage location.
77	MOV	" 404	initialization of overflow counter.
100	LDX	800,1,0	800 $\longrightarrow$ IR2; 0 $\longrightarrow$ IR1
101	CLA	500,1	500 the first time, incrementing by one in each loop.
102	TRC	400	compare to Z
103	TRU	107	$A < \alpha$ - here we get a count
104	TRU	112	$A > \alpha$ (In this case we find cumulative sum)
105	ADB	402,0,1	$A = \alpha$ - here we get a count
106	TRU	110	go to location which brings up next number
107	ADB	401,0,1	adding one to "less than" counter

```
110   TRX   101,1,1     go back and bring in next
                        number until 800th has been
                        examined.

111   HLT               stop after all numbers have
                        been examined

112   ADD   403,0,1     add previous sum, watching for
                        overflow

113   STR   403         store back into cumulative sum

114   SNR   116,0,100   if overflow, go to loc. 116

115   TRU   110         if no overflow, go to loc. which
                        examines next number.

116   ADB   404,0,1     since all values are positive
                        there can be no negative
                        overflow.

117   TRU   110
```

11. INPUT-OUTPUT SYSTEM

11.1 General Concepts of the Input-Output System -

a) The weakest point of any computer is its I-O system. Its timing may be measured in milli-seconds, whereas the timing of the rest of the computer may be measured in micro-seconds.

b) Data processing, such as the commercial work of insurance companies, etc., requires a vast amount of data handling (input), whereas scientific problems require little data handling but much internal work such as formula solving, etc. MOBIDIC was designed for efficient data handling.

c) Peripheral equipment is used to save main computer time. This equipment is "off-line", which means that it runs independently of the main computer. An example of its use would be converting cards to tape. It takes as much time on the off-line equipment as comparable equipment connected with the computer, but it does not use valuable computer time.

d) A buffer is equipment in the computer which assembles incoming or outgoing information until it is ready for main computer access.

e) Types of input and output media are:

1) Magnetic tape

2) Punch cards

3) Paper Tape

These are considered forms of external storage. In general, this information is <u>not</u> addressable for individual datum as is the information in storage.

f) Types of input and output devices:

1) Magnetic tape units, which are capable of reading and writing on magnetic tape

2) Punch card reader and punch

3) Paper tape reader and punch

4) Flexowriter

11.2 <u>Constituents MOBIDIC Input-Output System</u>

11.2.1 <u>Constituents</u>:

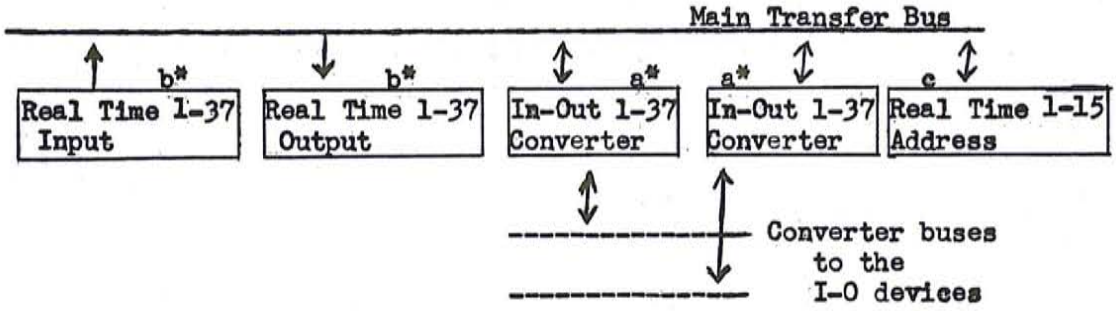
a) In-Out Converters

b) Real Time Input and Output Registers

c) Real Time Address Register

d) Addressable flip flops in main machine e.g., ISN, NHC

e) Error flip flops and controls



* no parity bit; 37th bit is the sign bit.

11.2.2 General Principles:

a) The converter(s) synchronize(s) operation of I-O devices with the main machine.

b) Converters may operate simultaneously and concurrently with the main program except that the main program will be interrupted at suitable times to allow memory access. This requires 8 μsec and is not under programmer control. However, the basic timing cycle allows for this memory access time. Therefore, MOBIDIC can read, write and compute simultaneously.

c) An I-O order selects the specified I-O device and a free converter. If the I-O device is in use, or no converter is free, then the main machine cycle stops until the device and/or the converter becomes free.

Any converter can be connected to any device. When the I-O operation is complete, the device is disconnected from the converter and the converter, in turn, is disconnected from the central machine. Another I-O order, if present, will be picked up only when the converter is disconnected.

11.2.3 Magnetic Tape Characteristics (subject to change):

a) The tape speed is 150 inches per second.

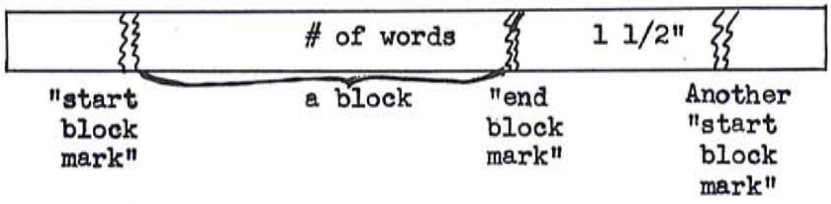
b) Characters are spaced on tape at .0036" intervals.

c) Characters are written or read on tape at 24 μ sec. intervals.

d) Characters are recorded on magnetic tape in eight parallel channels. A channel is a column of holes and/or spaces lengthwise on the tape. Six of the channels correspond to the alphanumeric character code, one is a parity bit for each character, and the last is used for control purposes.

e) Information is stored in units called "blocks", normally consisting of several words. There is no minimum or maximum number of words per block. This is subject only to the length

of the tape itself. The concept of "block" applies to mag tape. The beginning and the end of a block are indicated by characters called "start block marks" and "end block marks":



11.2.4 Characteristics of Paper Tape, Punch and Reader, and Flexowriter:

a) Normal widths of paper tape are 7/8", 11/16".

b) The number of channels on the tape differs according to tape width and equipment. Normally, 5,6,7, or 8 channels are used.

c) The Flexowriter can type or type and punch at the rate of 10 characters per second, which, by comparison to other output devices, is very slow. The Flexowriter is not an input device.

d) The Paper Tape Reader reads 270 characters or less per second, depending on the device itself.

e) The Paper Tape Punch punches 60 characters per second.

11.2.5 Word and Character Structure:

a) A Character indicates a configuration of 1, 3, 6 or 8 bits.

1) A 1 bit character indicates a sign representation.

2) A 3 bit character indicates an octal representation.

3) A 6 bit character usually indicates an alpha-numeric representation.

4) An 8 bit character is the Fieldata representation for the 1 bit character sign representation, the 3 bit character octal representation, or the 6 bit character alpha-numeric representation. The 7th bit is used as a signal specifying data or control, and the 8th as a parity bit.

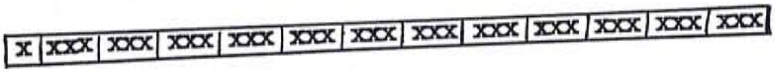
b) Words may consist of 1, 6, 7 or 13 characters.

1) One 6 bit character - (used mostly as a control word for in-put data)

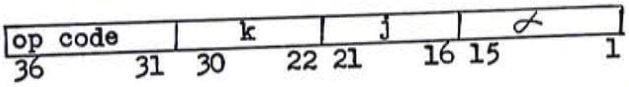
2) Six 6 bit characters -

3) Six 6 bit characters plus character for the sign bit -

4) Twelve 3 bit characters plus a character for the sign bit -



11.2.6 MOBIDIC In-Out Order Word:



α (15bits) designates either a memory address or one of the following addressable registers: A, B, Q, PC, PCS, or the converter being used.

α is automatically incremented by one after every memory access except when an addressable register is specified. i (6 bits) specified the in-out device address to be used during the instruction. Numeric codes for the devices have not yet been determined; therefore, the following abbreviations will be used: MT1...MT8, PTP, PTR, FLX. On MOBIDIC A, there are two paper tape readers, a 5 channel reader and an 8 channel reader, and two

paper tape punches (5 and 8 holes)

Since j is 6 bits long, the machine has the present capacity of using 63 (2^6-1) devices.

k (9 bits) indicates the number of words or blocks of words involved in the in-out operation, depending on the order.

Note: If the Op code and device are incompatible, there will be a failure.

12. MOBIDIC I-O ORDERS

12.1 WOK, ROK, WAN:

WOK kjα - Write Octal - Applies to PTP and FLX only. Words consist of 13 characters, the first character indicating the sign. The converter processes the sign bit and each subsequent 3 bit character to produce the 8 bit configuration corresponding to Fieldata code plus parity and control bit. A maximum of 511($=2^9 - 1$) words may be specified. The WOK order is not consistent if zero words are specified or if a device other than an output type device is specified.

WOK writes k words starting in memory position α onto device j. For further information refer to Table 1.

ROK kjα - Read Octal - Applies to PTR only. Words consist of 13 characters, the first indicating the sign (0 for plus, 1 for minus). Only the eight characters indicated by Fieldata code for 0 - 7 are legitimate. If any non-legitimate characters appear within a word, an alarm is caused; if it occurs elsewhere it will be ignored except that a stop symbol will terminate the operation. If a stop symbol appears in the middle of a word it will also cause an input-output alarm.

The three least significant bits are transferred to memory, except for the sign character which is converted to a single

bit. The first of the 13 characters must be the Fieldata code for a 0 or 1 (the sign) or else there will be an alarm.

This instruction reads k words from device j into memory starting at position α. The maximum number of words one can read without modification is $(511)_{10}$. ROK is further explained in Table 1.

Example: Write 100g words starting at location 500g on paper tape. Print the square root of each of these numbers on the Flexowriter. Assume the entrance to the Square Root routine is at 700g and it is entered with the argument in the accumulator and exits with the result in the accumulator.

One Method

```
20)  WOK   500g, PTP, 100g
21)  IDX   100g, 1, 1          100g ---> I^2, 1 ---> I^1
22)  CLA   477,1               C(500) ---> Acc
23)  TRL   700,4,0             Square Root Routine starts
                               in 700
24)  STR   177g, 1
25) [WOK   200g, FLX, 100g]
26)  MOV    31g, 25g           Put (TRU 27) in 25
27)  TRX    22,1,1             I^2 - 1 ---> I^2, I^1 + 1 ---> I^1
30)  HLT
31)  TRU    27                 Dummy Order
```

Note that we have plenty of time to compute the square root

because the Flexowriter is so slow. This program will not work with magnetic tape because the magnetic tape is faster and the computation could not be completed in time.

ISN Flip-Flop

ISN flip-flop is addressable (0102) and may be set by means of the SNS instruction preceding an I-O order. It is automatically reset to Q immediately after the I-O order is sent to the converter.

ISN is set to 1 to indicate "Interpret Sign Mode". The exact significance of interpret sign mode is discussed for each applicable order. Basically, it has the effect of adding a character to each word to indicate the sign. The character is a 1 or 0 in Fieldata code to indicate negative or positive data. If ISN is not set, the sign is automatically set to 0.

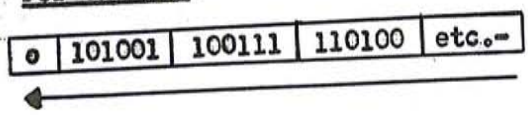
WAN k ---→ jα - Write Alphanumeric - Applies to PTP, FLX, MT. This operation writes k words on output device j. The first word comes from α and successive words from succeeding positions $\alpha + 1$, $\alpha + 2$, etc., unless α is an addressable register, in which case the effective address does not change.

If a magnetic tape unit is specified, suitable block marks are automatically inserted in the appropriate places.

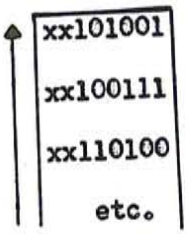
Word formation depends upon the state of the ISN flip-flop:

a) When the ISN is not set, six 6-bit characters are formed with bits 31-36 recorded first, bits 25-30 next, etc.

For example:

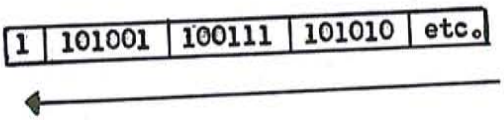


would on tape look like

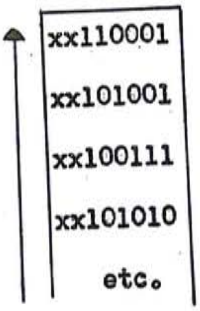


b) When the ISN is set, seven 6-bit characters are formed with the sign bit as the first character on tape, then bits 31-36, etc.

For example:



would on tape look like



WAN is explained further in Table 1.

Example: Write 322 words in alphanumeric from locations 100_8 ff on MT1 with sign and on PTP without sign.

```
500)  SNS   501,0,0102
501)  WAN   100₈, MT1, 322₈
502)  WAN   100₈, PTP, 322₈
```

Write the same words from the same locations on MT1 with sign and on PTP with sign.

```
500)  SNS   501,0,0102
501)  WAN   100_8, MT1, 322_8
502)  SNS   503,0,0102
503)  WAN   100_8,PTP,322_8
```

Write the same words from the same locations on MT1 without sign.

```
500)  SNR   501,0,0102
501)  WAN   100_8, MT1, 322_8
```

Homework:

Given numbers in positions 100_8 - 1000_8. Write the contents of 100_8, 102_8.....1000_8 on paper tape in octal. Write the contents of 101_8, 103_8.....777_8 on MT1 with sign and on the Flexowriter in alphanumeric without sign.

```
21   LDX   341,1,2      2 ---> I1, 341 ----> I2
22   CLA   76,1         100 ---> Acc
23   STR   2000
24   ADB   23,0,1
25   TRX   22,1,2
26   WOK   2000,PTP,341
27   LDX   340,1,2      2 ---> I1, 340, ---> I2
30   CLA   77,1         100 ---> Acc
31   STR   2500
32   ADB   31,0,1
```

```
33      TRX     30,1,2
34      SNS     35,0,0102       Sets flip-flop
35      WAN     2500,MT1, 340
36      WAN     2500, FLX, 340
37      HLT
```

The RAN instruction is explained in Table 2.

TABLE 1.

Reference Table for WAN, WOK, ROK.

ORDER	TYPE OF ORDER	DEVICE	ISN FF	NO. OF CHARACTERS PER WORD	WORD STRUCTURE	9th BIT OF K	AMOUNT OF INPUT OR OUTPUT	COMMENTS
WAN	Write	Mag. Tape P.T.Punch Flexo-writer	1	7	A six bit character sign followed by six 6-bit characters of word with bits 31-36 recorded first.	Affects value of k only	k words	Writes terminating block mark when finished. Is inconsistent if zero words are specified or if an input type device is specified.
			0	6	Six 6-bit character ignoring sign, with bits 31-36 first and so on.	Same	Same	Same
WOK	Write	P.T.Punch Flexo-writer	0 or 1	13	Sign digit followed by 12 octal digits. Each octal digit is converted to equivalent 6-bit form.	Same	Same	Same as above. No need for ISN FF. The sign is automatically interpreted.
ROK	Read	P.T. Reader	0 or 1	13	13 Octal digits	Same	Either k words or all words preceding a Stop Code, depending on which comes first. The Stop Code is not transferred to memory.	Is inconsistent if zero words are specified or if a device other than an input device is specified. Will fail if a character other than "0" through "7" appears in the middle of a word. Such a non-legitimate character is ignored when it is at the end of a word.

12.2 RAN k j - Read Alphanumeric - Applies to magnetic tape and paper tape

TABLE 2.

Reference Table for RAN:

TYPE OF ORDER	DEVICE	ISN FF	NO. OF CHARACTERS PER WORD	WORD STRUCTURE	9th BIT OF K	AMOUNT OF INPUT OR OUTPUT	OTHER COMMENTS
Read	Mag. Tape	1	7	Same as WAN	0	k (mod 2^8) blocks.	Searches for beginning of block mark before reading. Stops at end of block even if this is < k words. Goes to end of block if k < number of words in block.
		0	6	Same as WAN	1	k (mod 2^8) blocks	Same as above.
					0	k words	
	Paper Tape Reader	1	7	Same as WAN	Affects value of k only	k words or until Stop Code is encountered.	If Stop Code occurs in the middle of a word, there is an I-O alarm. It must occur after a full word, and it is not read into memory.
		0	6	Same as WAN	1	k words or until Stop Code is encountered.	The Stop Code may occur after any number of characters. If necessary, the low order bits of the last word will be filled with zeros and then the order will be terminated. Since it will read only one word regardless of what k says, it can be included in a loop. The Stop Code is read into memory as is any other order.
		0	1	One 6-bit character in low order position. Rest of word is 0.	0	One word only regardless of k.	

NOTE: If you have written in the ISN mode, be sure you read in the ISN mode. The opposite is also true.

Examples:

Read 300_8 words from MT1 with sign into location 1000_8 and following.

```
25)  SNS   26,0,102          sets ISN flip-flop
26)  RAN   1000,MT1,300
```

Read 25_8 blocks from MT1 with sign into location 500_8 and following.

```
50)  SNS   51,0,102
51)  RAN   500₈,MT1,425₈
```

Read 255_8 blocks in the same manner.

```
50)  SNS   51,0,102
51)  RAN   500,MT1,655₈
```

Read 200_8 words with sign from MT1 into location 500_8 and following.

Write all positive numbers consecutively on MT2 without sign.

CAUTION: Unless all positive numbers are consecutive, they will not be stored consecutively on MT2, because spaces will be left for negative numbers.

```
24   MOV   70000,10          0 ---> 10
25   SNS   26,0,102          Set ISN
26   RAN   500₈MT1,200₈
27   SEN   27,0,[IOD] MT1    Remain at (27) as long at MT1 is busy
28   LDX   200,1,0           200 ---> I², 0 ---> I¹
29   CLA   500,1
30   TRN   33
31   STR   1000,1
32   ADB   10,0,1
```

```
33   TRX   29,1,0
34   LOD   11,,0011          MSK ---> Q Register
35   CLA   10
36   SHL   21
37   MSK   40
40   WAN   1000 , MT2, [  ]
11)  007770000000
```

12.3 RRV, SKP, BSP, RWD, WWA.

RRV k, j - Read Reverse - Applies to magnetic tape units only.

This operates in the same manner as the RAN order except that reading takes place in the reverse direction. However, the sign bit appears in its proper place.

Example: if the characters A_{sn}, A_1, A_2, A_3, A_4, A_5, A_6 were recorded in that order in the forward direction, then when RRV is used, the word will be assembled as $A_{sn}, A_6, A_5, A_4, A_3, A_2, A_1$.

SKP k, j - Skip - Applies to magnetic tape units only.

The tape searches in a forward direction and counts one block skipped for each set of end block marks. It can skip a maximum of 511 blocks. Skipping x blocks takes as much time as reading x blocks.

BSP k, j - Backspace - Applies to magnetic tape units only.

The tape searches in a reverse direction and counts one block backspaced for each set of start block marks. It can backspace a maximum of 511 blocks.

RWD j - Rewind - Applies to magnetic tape units only.

Rewinds magnetic tape j. The instruction energizes the high speed rewind control switch at the magnetic tape unit. The converter is then disconnected from the device and the rewinding operation proceeds independently. The tape is rewound completely to the beginning. If an in-out order causes the tape unit to be again connected to a converter the tape is fully rewound.

WWA k, j - Rewrite Alphanumeric - Applies to magnetic tape units only.

This searches in a forward direction for beginning block marks and then writes k words. ISN mode may or may not be used. The existing set of start block marks is left undisturbed but a new set of end block marks is inserted in the appropriate places.

WWA is used to replace information stored on a given length of magnetic tape with new information without disturbing the remainder of the tape. If the new information is allowed to occupy too much space on the tape, one or more blocks may inadvertently be destroyed. If the new block occupies less space than the block being written over, it may not completely erase the old block.

Two restrictions --

1) The number of rewritten words must contain a total number of characters equal to the number of characters in the original block within one

word length, the reason being that it takes tape time to accelerate and decelerate. In writing too few words, the tape may not accelerate properly, and the channels may not be lined up in the proper manner.

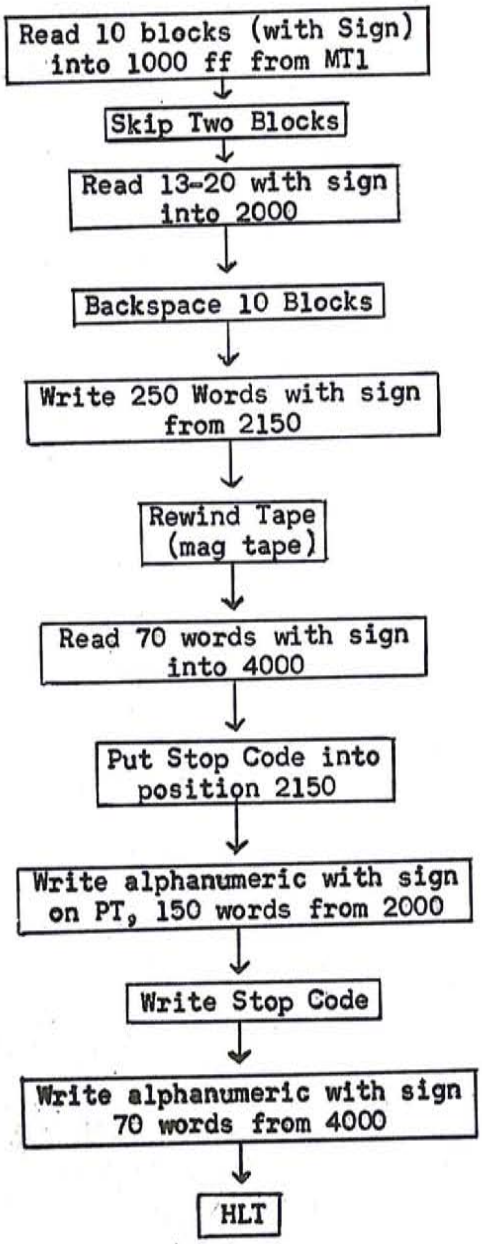
2) A block may not be rewritten if it contains more than 511 words of 7 characters each (i.e. 3577 characters).

Problem:

Given $(20)_{10}$ blocks of $(50)_{10}$ words each, with sign, on MT1, and 70 words in Alphanumeric, with sign, on PT.

Read 1st 10 blocks into positions 1000 and following, with sign, write blocks 16-20 with sign in the same place on tape as blocks 11-15. Put contents of blocks 13-15 on P.T. with sign followed by stop code. Then put out the 70 words from the 1st PT following this.
Rewind Tape.

Flow Chart


Read 10 blocks (with Sign) into 1000 ff from MT1
Skip Two Blocks
Read 13-20 with sign into 2000
Backspace 10 Blocks
Write 250 Words with sign from 2150
Rewind Tape (mag tape)
Read 70 words with sign into 4000
Put Stop Code into position 2150
Write alphanumeric with sign on PT, 150 words from 2000
Write Stop Code
Write alphanumeric with sign 70 words from 4000
HLT

(All the following coding uses octal memory locations but decimal numbers elsewhere.)

```
100  SNS  101,0,102          Set ISN Flip-Flop
101  RAN  1000,MT1, 266(10)  Reads 10 blocks
102  SKP  0,MT1,2            Skip 2 blocks
103  SNS  104,0,102          Make sure ISN Flip-Flop is set
104  RAN  2000,MT1,264(10)   Reads 8 blocks
105  BSP  0,MT1,10           Backspace 10 blocks
106  SNS  107,0,102          Make sure ISN Flip-Flop is set
107  WAN  2150,MT1,250(10)   Write blocks 16-20
110  RWD  0,MT1              Rewind
111  SNS  112,0,102          Make sure ISN Flip-Flop is set
112  RAN  4000,PTR,70(10)    Read 70 words into 4000
113  MOV  77,2150            In location 77 is the stop code
114  SNS  115,0,102          Make sure ISN Flip-Flop is set
115  WAN  2000,PTP, 151(10)  151 = 3 blocks (13-15) and stop code
116  SNS  117,0,102          Make sure ISN Flip-Flop is set
117  WAN  4000,PTP,70(10)    Write out paper tape
120  HLT
```

Homework:

Read 200_8 words with sign from MT1 into location 500 and following. Write all positive numbers on MT2 without sign and all negative numbers on MT3 with sign.

```
 76  CLA  70000
 77  STR  70002      } Initialize I2 and I3
100  STR  70003
```

101	SNS	102,0,0102	Set ISN Flip-Flop
102	RAN	500_8,MT1,200_8	Read 200 words into 500 Flip-Flop
103	SEN	103,0,IODMT1	Hold up until finished
104	LDX	200_8,4,0	200 ---> I^1, 0 ---> I^4
105	CLA	500,4	Examine first word, then second, etc.
106	TRN	113	
107	STR	1000 ,3	If positive, store in 1000
110	ADB	70003,0,1	Increase I^3, so next positive number will be stored in 1001, etc.
111	TRX	105,4,1	Go back and examine next number, when finished
112	TRU	116	Go to 116
113	STR	2000,2	If negative, store in 2000
114	ADB	70002,0,1	Increase I^2, so next negative number will be stored in 2001, etc.
115	TRU	111	Go back and examine next number, when finished
116	CLA	70003	Put positive counter into Accumulator
117	SHL	21	Shift into k position
120	ADD	132	Add dummy
121	STR	122	Store in write order
122	WAN	1000_8,MT2, [k]	Write positive numbers on MT2
123	CLA	70002	Put negative counter ---> Acc
124	SHL	21	Shift into k positive
125	ADD	133	Add dummy
126	STR	130	Store in write order

```
127  SNS  130,0,0102          Interpret sign
130  WAN  2000₈,MT3,[k]       Write negative numbers on MT3
131  HLT
132  WAN  1000₈,MT2,  o---o
133  WAN  2000₈,MT3,  o---o
```

13. MOBIDIC CONVERTER

13.1 Converter Instruction Details:

The Converter is a 36 bit register divided as follows:

Bits 1-15 (α) -- ADC (Address Counter Register). This is automatically sequenced by a count of one after every memory access except when an addressable register is specified.

Bits 16-21(j) -- DAR (Device Address Register). This specifies the in-out device to be used during the instruction.

Bits 22-30(k) -- WBC (Word Block Register). This specifies the number of words or blocks involved in the in-out operation.

Bits 31-36 (op code) -- ISR (Instruction Register). This specifies the particular in-out order to be performed.

These four registers form a unit called the Converter Instruction Word (CIW), which is addressable.

Restrictions:

1) The contents of the Device Address Register cannot be modified.

2) The individual registers in themselves are not addressable, but can only be obtained by addressing the Converter Instruction Word.

The transfer of data between the converter and the central machine occurs, 37 bits in parallel, through a converter register called the word buffer (BFR). When the word buffer requires access to memory, a busy bit flip-flop is set.

Register Address	Register
70030	Converter Instruction Reg. #1
70031	Converter Instruction Reg. #2
70015	Converter being used

When the octal register address 70015 is used as the address of the input instruction it tells the converter to store the contents of its Buffer Register into its Instruction Word Register. The device address of the converter instruction word register is not included in the data transfer from the Buffer Register.

Only input instructions have this feature.

13.2 Order Modification

Timing is very critical.

1) The ISR may be effectively modified only when using

magnetic tape, when the number in WBC = 0, and when the RRV instruction is not involved.

a) If other than magnetic tape, an ISR change is useless since the converter is automatically disconnected as soon as WBC = 0, and therefore there will not be time for the modification procedure. If in the magnetic tape mode an order modification is sensed, the normal shut-off procedure is overlooked to permit the new order to be executed.

b) If the ISR is modified before WBC = 0, the converter will immediately attempt to execute the new order and the original order may not be suitably terminated. Neither the old nor the new order will be properly executed.

c) If there is a change to or from an RRV, there must be a number <u>other than 0</u> in WBC in order for it to work properly.

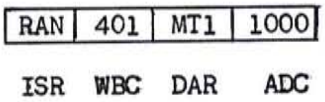
2) The DAR can never be modified. Changes in the DAR are inhibited by the hardware.

3) The WBC and ADC should be modified only when the number in WBC ≥ 1. Even if the number in WBC is ≥ 1, it still will not work properly if timing is ignored. One or more memory locations may not be loaded as intended.

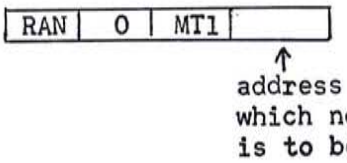
Converters are assigned in order. The first I-O order goes to Converter #1, the 2nd to Converter #2 assuming Converter #1 is still busy, and the 3rd I-O order goes to the first converter to be free. If both converters are free at the same time, the 3rd I-O order will go to Converter #1 since the I-O order is sent to the lowest numbered converter which is free.

Example: Determine how many words are in a block which was read from MT1 into location 1000_8 and following. Assume Converter #1 is used.

10) RAN 1000_8, MT1, 401_8 Read one block: at start



11) SEN 11, 0, IODMT1 At finish



12) MOV 70000, 21

13) CLA 70030 C(CIR1) ---> Acc

14) RPA 21 Put C(ADC) in 7

15) CLA 21 Assumes ADC is address of next position.

16) SUB 20 Now the number of words desired are in the accumulator

17) HLT

20) + 1000

21) o ---- o ADC Temporary storage

Example: Determine whether Converter #1 contains a write order:

```
10)  MOV   70000,1000     Put 0 → 1000
11)  LOD   1001,, 0011    Put mask in Q
12)  CLA   70030          C(CIR1) --→ Acc
13)  MSK   1000           Puts op code in 1000
14)  CLA   1002           C(1002) --→ Acc
15)  TRC   1000           Compare to instruction code
16)  TRU   21             A < ∝; may have a write order
                          (either write or rewind 77)
17)  TRU   out            A > ∝; do not have write order
20)  TRU   out            A = ∝; do not have write order
21)  CLA   1001           0770——0 ---→ Acc
22)  TRC   1000           Compare to instruction code
23)  TRU   Error          A < ∝; not a write order (it's impossible
                          to go here, because 77 is the highest code)
24)  TRU   in             A > ∝; this is a write order
25)  TRU   out            A = ∝; rewind order, not write order

1001)0770--0

1002)0730--0
```

Example: Write 999_{10} words from location 200_{10} and following as one block without sign. Assume Converter #1 is used.

```
    10)  0007770 - 0
    11)  0007510 --- 0
    12)  0000020 --- 0
START→20)  WAN   200,MT1,777₈   Put maximum number into WBC
    21)  CLA   70030            Converter #1 --→ Acc
    22)  TRC   70030            Compare to be sure WBC has changed
    23)  TRU   26               Acc<∝; has changed
    24)  TRU   26               Acc>∝; has changed
    25)  TRU   22               Acc = ∝; has not changed yet
    26)  LGM   10               WBC --→ Acc
    27)  TRC   12               TRC to 2, because while Acc may be 2,
                                converter is down to 1
    30)  TRU   22               Acc < ∝
    31)  TRU   22               Acc > ∝
    32)  LOD   10,,0011         Acc = ∝; load mask
    33)  CLA   11               Remainder of words --→ Acc
    34)  MSK   70030            Put remainder of words in converter
```

Class Problem:

Write the contents of locations 1000 - 4000_8 as one block on MT1 without sign. Assume Converter #1 is used.

```
    11)  0007770 ---- 0
    12)  0007770 ---- 0
    13)  0007770 ---- 0
    14)  0000040000
    15)  + 2
START-→17)  LDX   3,1,0          3 ---→ I2, 0 ---→ I1
    20)  WAN   1000,MT1,777₈     Write 777₈ words from 1000 on
                                 magnetic tape
    21)  CLA   70030             Converter #1 ---→ Acc
    22)  TRC   70030             Compare to itself to assure WBC
                                 change before computation and not
                                 during
    23)  TRU   26                Has changed, continue
    24)  TRU   26                Has changed, continue
    25)  TRU   22                Has not changed - go back
    26)  LGM   11                WBC ---→ Acc
    27)  TRC   15                Compare to 2 because while Acc may
                                 be 2, converter may be down to 1
    30)  TRU   22                Acc <∝ not yet 2
    31)  TRU   22                Acc >∝ not yet 2
    32)  CLA   12,1              Acc  =∝  takes the next WBC
    33)  LOD   11,0011           Load mask
    34)  MSK   70030             New WBC ---→ Converter #1
    35)  TRX   21,1,1            Start over
    36)  HLT
```

Example: Assume first word on tape is to convey the address of the data in the block following it. Read block into memory. Assume the first word is not zero.

1st Method

11)	CLA	70000	Clears Acc
12)	RAN	$70010,MT1,401_8$	Reads the block into Acc
13)	TRZ	13	Waits for the first word
14)	RPA	70030	Puts address of first word in converter
15)	HLT		

2nd Method

Assume 1st word is of the form RAN address, MT1, 401_8

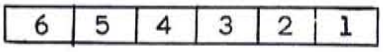
11) RAN $70015,MT1,401_8$

Method which does not work

RAN	$70010,MT1,401_8$	
CLA	70010	
RPA	70030	Too fast, i.e., won't have first item on tape in accumulator.

Problem:

There is a word in the accumulator consisting of six 6-bit characters:



Reverse the characters so that they are in ascending order and do this in 140 μs or better.

CYS 6 — 5 4 3 2 1 6

STR k

CYS 12 — 3 2 1 6 5 4

LOD MSK,,Q — Mask = o-o 1-1 o-o o-o 1-1 0-o

MSK k — 5 2 3 2 5 6

SLL 6 — 2 1 6 5 4 o-o | 1-1 o-o o-o 1-1 o-o o-o

CYS 6 — 1 6 5 4 o-o 2

MSK k — 1 2 3 4 5 6

Example: Read 4 blocks from MT2 into location 2000 and following and then 500 words without sign from location 200 and following on magnetic tape 2. Assume Converter #1 is used.

```
     5)  0000020----0
     6)  0003770----0                 3 instead of 7 because blocks are
                                      being read
     7)  0777770077777₈
START->11)  RAN   2000,MT2,260₁₀      Read 4 blocks; 260₁₀ = 404₈
    12)  CLA   70030                  Converter #1 ---> Acc
    13)  TRC   70030                  Compare to itself to assure WBC
                                      changes before computation and not
                                      during.
    14)  TRU   17                     A < ∝ ; has changed
    15)  TRU   17                     A > ∝ ; has changed
    16)  TRU   13                     A = ∝ ; has not changed
    17)  LGM   6                      Puts WBC in Acc
    20)  TRC   5                      Compare to 2
    21)  TRU   13                     A < ∝ ; not yet 2
    22)  TRU   13                     A > ∝ ; not yet 2
    23)  LOD   7,,0011                A = 2 ; load mask
    24)  CLA   27                     Dummy order ---> Acc
    25)  MSK   70030                  New order into converter
    26)  HLT
    27)  RAN   200,0,500              Dummy order
```

Example: Read 1 block without sign from MT1. When a word equal to the contents of location 100 is encountered, stop reading and rewind the tape. Assume C(100) are not 0.

```
11)  CLA   70000
12)  RAN   70010,MT1,401₈
13)  LOD   23,,70011
14)  TRC   100
15)  TRU   14
16)  TRU   14
17)  CLA   70000          0 ---> Acc
20)  MSK   70030          0's ---> WBC
21)  RWD   0,MT1
22)  HLT
23)  0007770 ------0      Leaves original word untouched,
                          except puts 0's in WBC
```

14. MOBIDIC BASIC TIMING CYCLE

14.1 Timing Functions:

The basic cycle contains the logic which is common to all instructions. Each instruction also has its own instruction logic which happens simultaneously to the basic cycle. The time required to perform one cycle is normally 16 μ sec. Some instructions, such as multiplication and division, may take considerably longer. Each basic cycle is subdivided into 8 periods called timing functions, each timing function being normally 2 μ sec. long. For instructions requiring more than 16 μ sec., any of the 8 timing functions can be extended in increments of 2 μ sec.

The basic timing cycle and its operations are divided into two classes.

1) The first class consists of operations performed for the execution of the particular instruction involved.
2) The second class refers to basic timing cycle operations performed for all instructions.

The basic cycle in its timing functions is shown in detail in Table I of the Preliminary Programming Manual.

14.2 Busy Bits:

The transfer of data between the converter and the central machine occurs with 37 bits in parallel, through a

converter register called the word buffer. When the word buffer requires access to memory*, a busy bit (BB) flip flop is set. During each basic cycle, all the busy bits are sampled to determine whether or not memory access is needed. If so, the basic cycle is suspended until all busy bits are cleared. In the event two or more converters require access to memory simultaneously, the converters are given a priority as follows:

1) Real Time
2) Magnetic Tape
3) Others

Other than this, converters are taken in numerical order.

During certain long orders--in particular, Multiply, Divide, Shifts, Cycles, Normalize, Move--the busy bits are also sampled. If any are present, they are processed.

It takes 8 μ sec. for each memory access. However, no time is lost if memory access occurs during one of the above-mentioned orders. The longest time which can occur without handling a busy bit is 22 μ sec.

*Access to the registers A, B, Q, PC or PCS is also possible.

15. TRAPPING MODE

The trapping mode is used mainly in diagnostic procedures. It can be controlled either by the program or by the operator. The following discussion applies only to program control.

If the trapping mode flip-flop (TRA) is set, and the machine is given any of the instructions in the list below, it will not perform the instruction; it will instead take the next instruction from location 0, the contents of the Program Counter will be sent to the B register, and the trapping mode flip-flop will be reset.

TRL	TRN	SEN
TRS	TRP	SNS
TRX	TRZ	SNR

TRU itself can be trapped only when it contains a 0 in β_{16} and the trapping mode flip-flop has previously been set. The trapping mode flip-flop is not addressable. It is set and reset only through the use of the two low order bits of β in a TRU instruction. The reason for the following cases is to make the TRU order itself immune to the trapping mode:

Chart For Trapping Mode

TRU		
β_{16}	β_{17}	
1	0	If the Trapping Mode flip-flop = 0, take the next instruction from $\alpha+I\delta$ and set the TRA to 1.

		$\alpha+I\gamma$ goes to PC. If the Trapping Mode flip-flop is already 1, take the next instruction from $\alpha+I\gamma$ and leave the TRA set to 1. $\alpha+I\gamma$ goes to PC.
0	0	If TRA = 1, take the next instruction from location 0 and reset TRA to 0. The order is handled as any other transfer.
0	1	If TRA = 0, take the next instruction from $\alpha+I\gamma$ and leave TRA = 0. The order is handled as any other transfer.
1	1	If TRA = 1, take the next instruction from $\alpha+I\gamma$ and reset TRA = 0. $\alpha+I\gamma$ goes to PC.
		If TRA = 0, take the next instruction from $\alpha+I\gamma$ and leave the TRA set to 0. $\alpha+I\gamma$ goes to PC.

The trapping mode is used primarily for debugging. It permits one to store a small routine in position 0 which can be entered automatically on every transfer.

Example: As an illustration, consider that every time we get to a control transfer (except TRC) we would like to put out on paper tape in octal the contents of positions 100-125 and return to the main program.

```
58  TRU  59,0,1        Sets the trapping mode
59  SNS  60,-,-
60
```

```
 0  WOK  100,PTP,26
 1  STR   12           Because in ADB, the new sum replaces
                       contents of Acc.
 2  CLA  70012         Because in MOV, C(α) ⟶ β and α ⟶ Q
 3  STR   13
 4  MOV  70011,14      Because in ADB, original contents of
                       Acc ⟶ Q
 5  ADB  13,0,1        Increase by 1, the PC which was originally
                       in β
 6  RPA   11
 7  CLA   12           Replace original contents of Acc
10  LOD  14,0,0011     Replace original contents of Q
11  TRU  [   ]
```

If one actually wanted to execute this SNS in his program, besides its use for the trapping mode, he would have to:

1) Obtain the trapped instruction in the accumulator.

2) Store the address of the trapped instruction.

3) Put the trapped instruction in memory (somewhere in the trapping program).

4) Replace the α in the location where the trapped instruction is now stored by a value of use to the trapping program.

5) Execute the "new trapped order" (after restoring all registers assuming they have been saved).

6) Modify the α part of the TRU to the main program appropriately.

7) Execute the TRU to the main program after restoring registers.

For example:

TRA = 1

---

519)TRN 79

In the accumulator, in reference to outlined steps:

1) TRN 79.

2) Store 79 in temporary storage, say location 25; therefore in location 25 we have 79.

3) Store TRN 79 in location 15.

4) Change location 15 to read TRN 35.

5) Restore accumulator (all registers). Execute order in location 15, i.e., TRN 35.

6) If the accumulator is positive, put 520 in TRU to main program. If the accumulator is negative, put 79 in TRU to main program.

This is an interpretative program, which executes the original instruction other than the ordinary way.

16. REAL TIME SYSTEM

Real Time System permits connection to external devices such as teletype equipment or another MOBIDIC. Information can arrive or depart at any time, for example, at intervals of seconds or hours, etc. No converters are involved.

There exists a Kineplex, which is a communications terminal device which puts data from various media, such as a teletype device, into the computer. It conveys information at 300 characters per second. It receives information depending upon how fast the words can be assembled. It emits information depending upon how fast the device can handle the information.

The Real Time System consists of 3 registers, two of which are addressable, and four addressable flip-flops:

	RIR =	Real Time Input Register
(70021)	RAR =	" " Address "
(70022)	ROR =	" " Output "
(0104)	RPE =	" " Parity Error Flip-Flop
(0105)	ROBB =	" " Output Register Busy Bit Flip-Flop
(0106)	ROR_{38} =	" " Output Register Bit #38 Flip-Flop
(0101)	ROPI =	" " Output Program Interrupt Flip-Flop

16.1 The Input Register (RIR)

RIR is a 40 bit non-addressable register:

Bits 1-37 comprise the standard MOBIDIC word.

Bit 38 is a spare bit at present.

Bit 39 is a Busy Bit, which is set when memory access is needed.

Bit 40 is the Program Interrupt Bit. When this is set, it causes an unconditional transfer to 0 upon completion of the instruction during which the PI bit was set and C(PC)$\rightarrow\beta$.

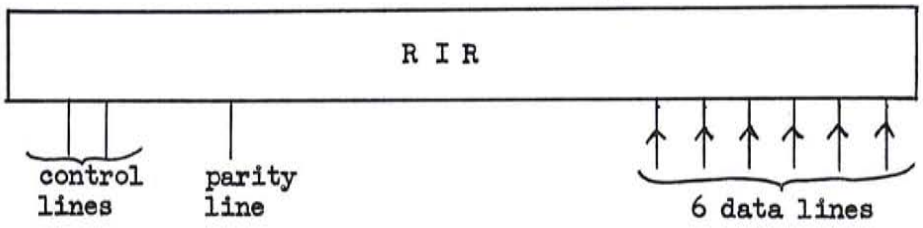
Note: A one in the PI (RIR_{40}) bit causes the same effect as the trapping mode except that the current order is trapped regardless of its operation code or β bit configuration.

The RIR has 9 lines, or bits, extending from it:

6 data lines

1 parity line to yield an odd number of 1's

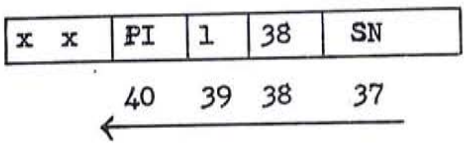
1 strobe line to feed signal in
1 ready line to feed signal out
} control lines



6 bits are received at one time. These comprise one character, or one frame. There are 7 characters per word. One character may be received every 6 μ sec and thus

a word can be received in 42 μ sec. Since there is a possible delay of 22 μ sec before memory access is allowed, it may take 64 μ sec between sending words to memory.

The first frame must look like this:



because there must be a 1 in the 39th bit position.

These bits are shifted into their correct positions when the rest of the word is filled with input data (6 other 6-bit characters). A 1 in the 39th bit (BB_9) signals the central machine that a memory access is required; the contents of RIR_1 - RIR_{37} are then transferred to the memory at the address specified by RAR, and RAR is incremented.

If the machine is "hung" (because of a halt or an I-O order which cannot be executed) and the PI bit is present, RIR will not accept any more information until PI is processed.

RIR is cleared when its contents go to memory.

Parity is checked on each set of incoming bits and a parity error causes the addressable flip-flop RPE to be set.

RPE can be cleared only by an SNR instruction from the program. RPE cannot be set by the program.

16.2 The Address Register (RAR)

RAR is a 15 bit register which is addressable (70021). The RAR specifies an address (in memory or of a register) for information coming from the RIR. It is automatically indexed after each memory access unless an addressable register is specified. The only addressable registers allowed are A,Q,B,PC,PCS. If 70015 is used with RAR, the effect is the same as using any other nonexistent register, i.e., a no-op.

Example: Keep track of the addresses of words containing parity errors. Assume fewer than 4096 words are being read, and also RPE = 0.

```
 7)  LDX  7777(8),1,0     Put 0 in I1, 7777 in I2
10)  LOD  17,0,0021       100---> RAR; this is the loc.
                          first read into, and this is
                          incremented by 1 with each new
                          loop.
11)  SNR  13,0,0104       Transfer if parity error
12)  TRU  11
13)  CLA  70021           C(RAR)---> Acc. if parity error
                          exists
14)  STR  1000,1
15)  TRX  11,1,1
16)  HLT
17)  +100
```

This program will be in a continuous loop until 4096 words with parity error are read in, since index register 2 is decreased by one only after a parity error is sensed.

16.3 The Output Register (ROR)

ROR is a 40 bit register which is addressable by any instruction which can store a word in it - e.g., STR, MOV, LOD, etc.

Bits 1-37 comprise the standard MOBIDIC word.

Bit 38 is a sensible flip-flop (0106) not used at present (ROR_{38})

Bit 39 is a Busy Bit. It is a sensible flip-flop (0105) called ROBB and is automatically set when information is sent to bits 1-37. When the last character is on the output lines it is reset. It can only be sensed and cannot be set or reset. Its programmer's use is to sense whether the word is still in ROR before loading another word. There is nothing mechanical to prevent the programmer from sending a word to ROR before it is cleared but if this is done incorrect results will be obtained. The time it takes to clear it is a function of how fast the receiver can take information.

Bit 40 is a sensible flip-flop (0101) called ROPI. Both ROR_{38} and ROPI can be set and reset by the program. Both must be set for each word transmitted because they may be different since they are not cleared when the word is used.

Note that since one of the reasons for having the real time system is to allow information to go from one MOBIDIC to another, we must be able to prepare the words going out in the manner needed for their reading. This is the reason we must have access to ROR_{38} and ROR_{40} (ROPI).

Example: Transmit 100_{10} words starting at 1000_{10}. Prepare the last word to interrupt the program when sent to another MOBIDIC.

```
 5   SEN  5,0,0105       Continue if ROR is not busy
 6   SNR  7,0,0101       Reset ROPI
 7   SNR  10,0,0106      Reset ROR38 (just so we know it's there)
10   LDX  99,1,0         0's --> I1, 99 --> I2
11   SEN  11,0,0105      Continue if ROR is not busy (part of loop)
12   LOD  1000,1,0022    Put word in ROR
13   TRX  11,1,1         Go to next line after 99 words
14   SEN  14,0,0105      Continue if ROR is not busy
15   SNS  16,0,0101      Set ROPI for last word
16   LOD  1099,1,0022    Put last word out with PI
17   SEN  17,0,0105      Wait till ROR is not busy
20   HLT
```

17. TABLE 3 - MOBIDIC SENSE FLIP FLOPS

Octal Address	Designation	Function	Console Switch	Can Be Cleared By Program	Can Be Set By Program
0000-0077*	IOD	In-Out Device Busy Signals	No	No	No
0100	OA	Overflow Alarm	No†	Yes	Yes
0101	ROPI	Real Time Output Program Interrupt	No	Yes	Yes
0102	ISN	Interpret Sign	Yes	Yes	Yes
0103	NHC	No Halt on In-Out Converter Error	Yes	Yes	Yes
0104	RPE	Real Time Parity Error	No	Yes	No
0105	ROBB	Real Time Output Busy Bit	No	No	No
0106	ROR_{38}	Real Time Output Reg. Bit #38	No	Yes	Yes
0110-0117	SFF1-SFF8	General Sense Flip Flops	Yes	Yes	Yes
0120-0127	SFF9-SFF16	General Sense Flip Flops	Yes	Yes	Yes
0130	IOA_1	In Out Alarm Converter 1	No†	Yes	No
0131	IOA_2	In Out Alarm Converter 2	No†	Yes	No
0135	TPE	Tape Erase	Yes	Yes	Yes
--	TRA	Trapping Mode	Yes	Yes**	Yes**

* These addresses correspond to the address of the device being sensed.

** Done only by β bits of TRU order or when an order is trapped.

† Can be reset by CLEAR ERROR BUTTON.

18. THE CONSOLE

The MOBIDIC Console is the external control unit for the MOBIDIC computer. It contains the control switches, indicators, and power switches required by the operators and the maintenance personnel to efficiently control the computer. The MOBIDIC Console consists of three panels. The top panel, shown in Figure 7-19, is not of use to programmers. The complete MOBIDIC Console is shown in the figure on the next page.

18.1 The Mode Controls (See Figure 7-21)

The mode controls consist of three push button switches which are electromechanically interlocked: The Run Switch, the One Cycle Switch, and the Single Pulse Switch.

18.1.1 Run Switch - This provides the gating levels necessary to operate the computer continuously after being started until stopped by the program, the operator, or an error.

18.1.2 One-Cycle Switch - This generates the gating levels necessary to stop the computer after one instruction is complete. This mode is useful in program checking and computer maintenance.

18.1.3 Single-Pulse Switch - This generates the gating levels necessary to gate "OFF" the normal pulse distribution system and to provide a single pulse for each operation of an initiation control. This mode is useful in computer maintenance.

Any one of these mode control buttons when pushed will release the other two.

18.2 The Initiating Controls (See Figure 7-21)

18.2.1 Start at Address Switch Register Switch - This switch initiates a program sequence where the location of the first instruction is specified by the contents of the Address Switch Register.

18.2.2 Start at Program Counter Switch - This switch initiates an instruction sequence where the location of the first instruction is specified by the contents of the Program Counter.

18.2.3 Program Read-In Switch - This switch initiates a special sequence which is used to read programs into the computer from punched paper tape. This requires two instruction words on a paper tape. The first is a READ which specifies an amount of data, the device, and the memory positions for the data. The second word address must be the position to which control is transferred after completion of the read-in. The remainder of the second order is irrelevant. All information must be in octal.

If the first word is not an I-O order, the machine will stall and can only be cleared by the

Emergency Halt button. If the first order is a write order, it will be executed.

18.2.4 Read-In Switch - This switch initiates a function which reads the contents of the Word Switch Register into the register or memory location specified by the contents of the Address Switch Register.

18.2.5 Read-Out Switch - This switch initiates a read-out operation which causes a memory location specified by the Address Switch Register to be read into the Memory Output Register. Then, the Memory Output Register may be displayed on the Bus Indicators.

18.2.6 Manual Instruction Switch - This swtich initiates the performing of the instruction in the Word Switch Register.

For manual read-in, set up the read order in the Word Switch Register and press this Manual Instruction Button. Then set up the address to which control is to be transferred in the Address Switch Register and switch the "Start at Address Switch Register Switch".

There are seventeen modes of operation for this computer. These include every combination of the Mode Controls and Initiating Controls with a single exception.

Examples - Run; Start at Address Switch Register

One-cycle; Program Read-In

Single Pulse; Manual Instruction Switch.

Exception - Single Pulse, Program Read-In

18.3 The Special Purpose Controls (See Figure 7-21)

18.3.1 Halt - The halt switch provides a manual method for stopping the computer. The computer will stop only after completing the instruction being executed, except that all input and output operations are completed before stopping. Other functions using the halt circuit to stop the computer are:

a) Emergency halt switch

b) Memory Parity Error

c) Overflow Alarm

d) Input-Output Alarm

e) Non-Existent Memory Address

f) Non-Existent Order Type

18.3.2 Emergency Halt - This switch provides a method of stopping the computer without waiting for the completion of the I-O instructions. It operates with the halt circuit, stops all the input-output devices, and stops all the converters.

18.3.3 Clear Memory - This function provides a method of resetting the contents of all memory locations to

zero. Normally, this is done before each new program is read into the computer. This function is initiated by a Clear Memory switch. Each memory will have an individual Clear Memory switch.

18.3.4 Clear Computer - This function provides a means of resetting most of the flip-flops in the computer. The only flip-flops not reset are the Halt flip-flop (which is set) and those in the timing function generator which are set at TF 4. The clear function is initiated by the Clear Computer Switch.

18.4 The Flip-Flop Controls (See Figure 7-21)

18.4.1 The Sense Flip-Flops (SFF^1-SFF^{16}) - These can be set or reset at any time. When locked from the console in either the set or reset position the programmer may only sense them. Use of the SNS or SNR orders will not change the state of the flip-flop.

18.4.2 The Sense and Control Flip-Flops - These can be set or reset by the operator only when the console is in the halt condition, since these flip-flops are interlocked with the halt circuit.

They include:

a) NHC - No halt on converter error. This is a sensible flip-flop.

b) ISN - Interpret Sign. This is a sensible flip-flop.

c) TRA - Trapping Mode.

d) TPE - Tape Erase. This is non-sensible but can be set or reset by the programmer. It operates only in conjunction with a write order and it erases either words or blocks as specified by k of the write order.

18.4.3 The Error and Alarm Flip-Flops - These have only a common reset control (Clear Alarm). They are:

1 - 7 MPE^n - Memory Parity Error (one for each memory)

1 - 4 IOA - In-Out Alarm (one for each converter)

OA - Overflow Alarm

NXM - Non-existent Memory Alarm

NXI - Non-existent Instruction Alarm

18.5 Console Displays and Switch Registers (See Figure 7-20)

18.5.1 Bus Indicator Register - Provides for a 38 bit display by which the contents of the following registers may be shown:

1. A register
2. B register
3. Q register
4. Program Counter Store
5. Memory Output Registers #1 - #7
6. Index Registers #1 - #4

Since the bus indicator circuits are connected directly to the main transfer bus, they will give an indication of actual computer operation. A special interlock circuit is provided to prevent the selected register from being read onto the bus while the computer is running.

18.5.2 Instruction Word Register - Displays the contents of the Instruction Word Register which is actually made up of the IR + G + X + AR registers.

18.5.3 Program Counter Indicator Register - Displays the memory address of the next instruction to be performed by the computer.

18.5.4 Address Switch Register - This is a 15-digit switch-register which can be used to provide register addressing during the Read-In and Read-Out operations.

18.5.5 Word Switch Register - This is a 37-digit switch register which is normally used in conjunction with the Read-In and Manual Instruction operations, but can also be used as an addressable "constant" register.

The switches and associated indicators of the switch registers are divided in groups of three to facilitate octal reading.

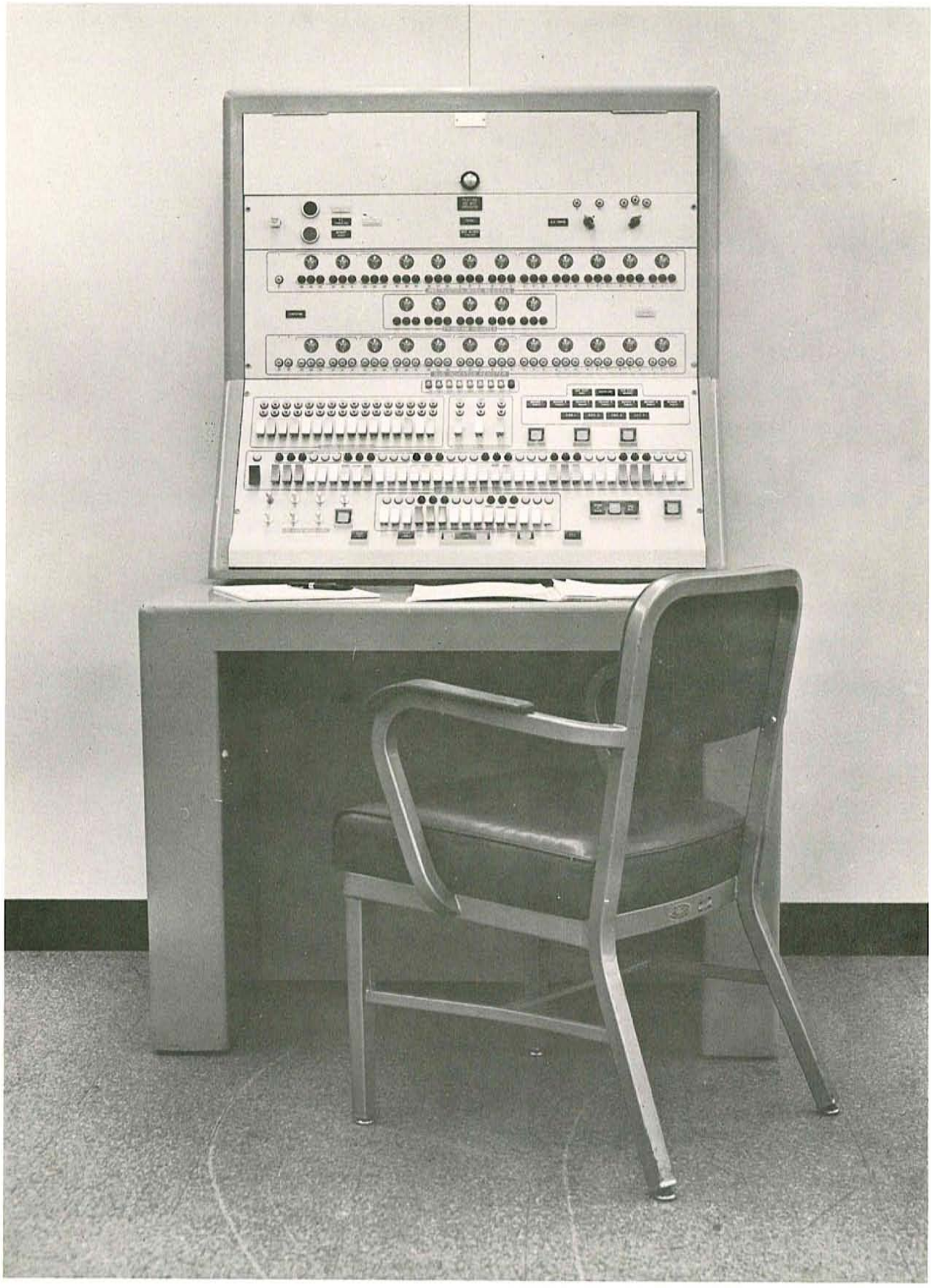
18.6 Special Controls and Indicators (See Figure 7-19)

18.6.1 Ready to Compute Indicator - When the machine is not computing (i.e. is in the Halt state), this light illuminates indicating that the computer is ready to compute, providing that the memories have power supplied to them.

18.6.2 Computing Indicator - When the machine is not in the Halt state the computing indicator illuminates.

18.6.3 No Error Halt Switch (KEW) - This is an alternate action push button switch. When the switch is "ON" the computer will not halt on conditions which set any of the Error Alarms. When the switch is "OFF" the computer will halt on any of the errors shown by the Error Alarms.

18.6.4 Disable Program Interrupt - This is a toggle switch which, when "ON", prevents the bit from the Real Time Register from interrupting the program. The associated indicator is illuminated.



COMPUTER CONSOLE

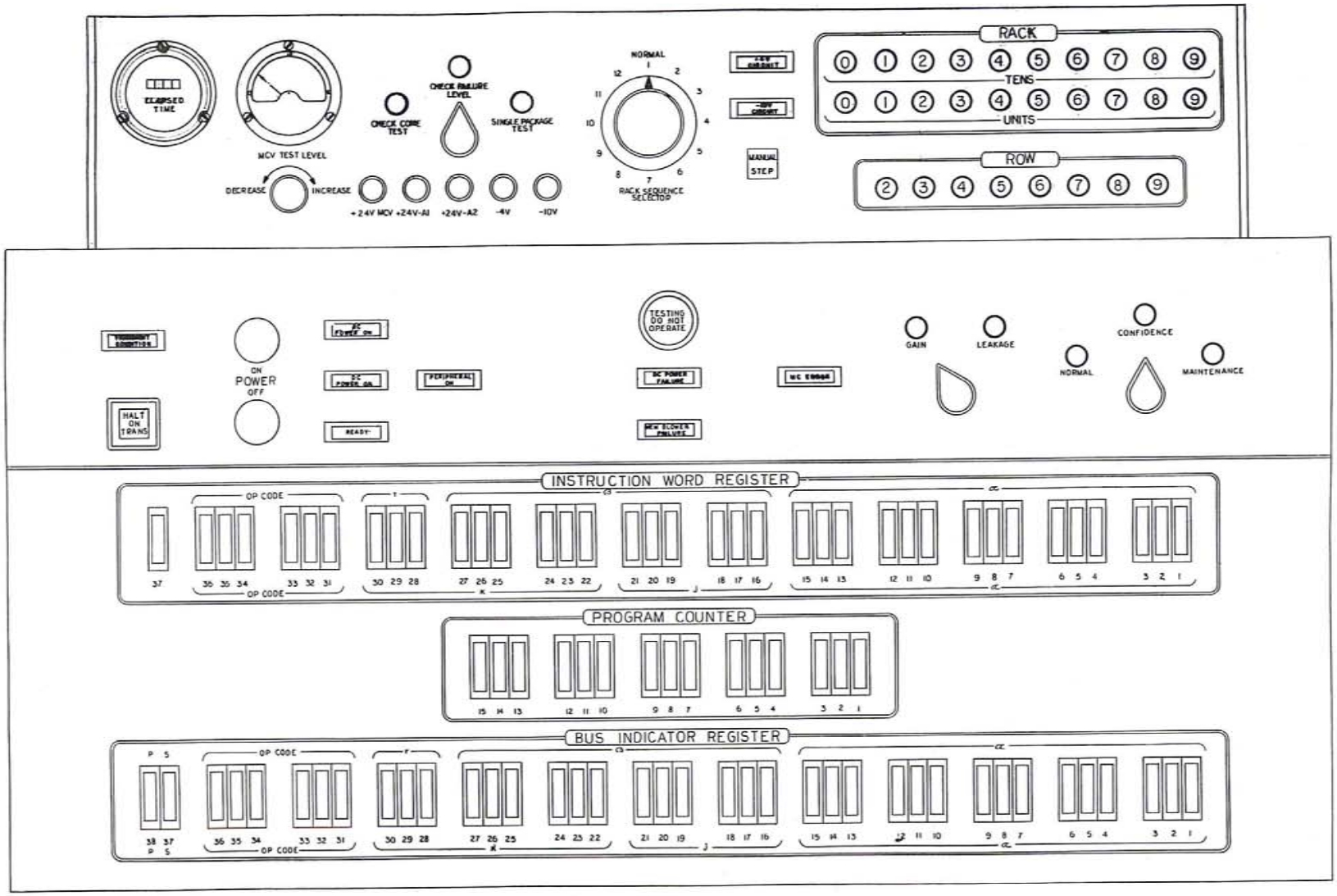


FIGURE 7-20 MARGINAL CHECK CONSOLE PANEL AND DISPLAY PANEL

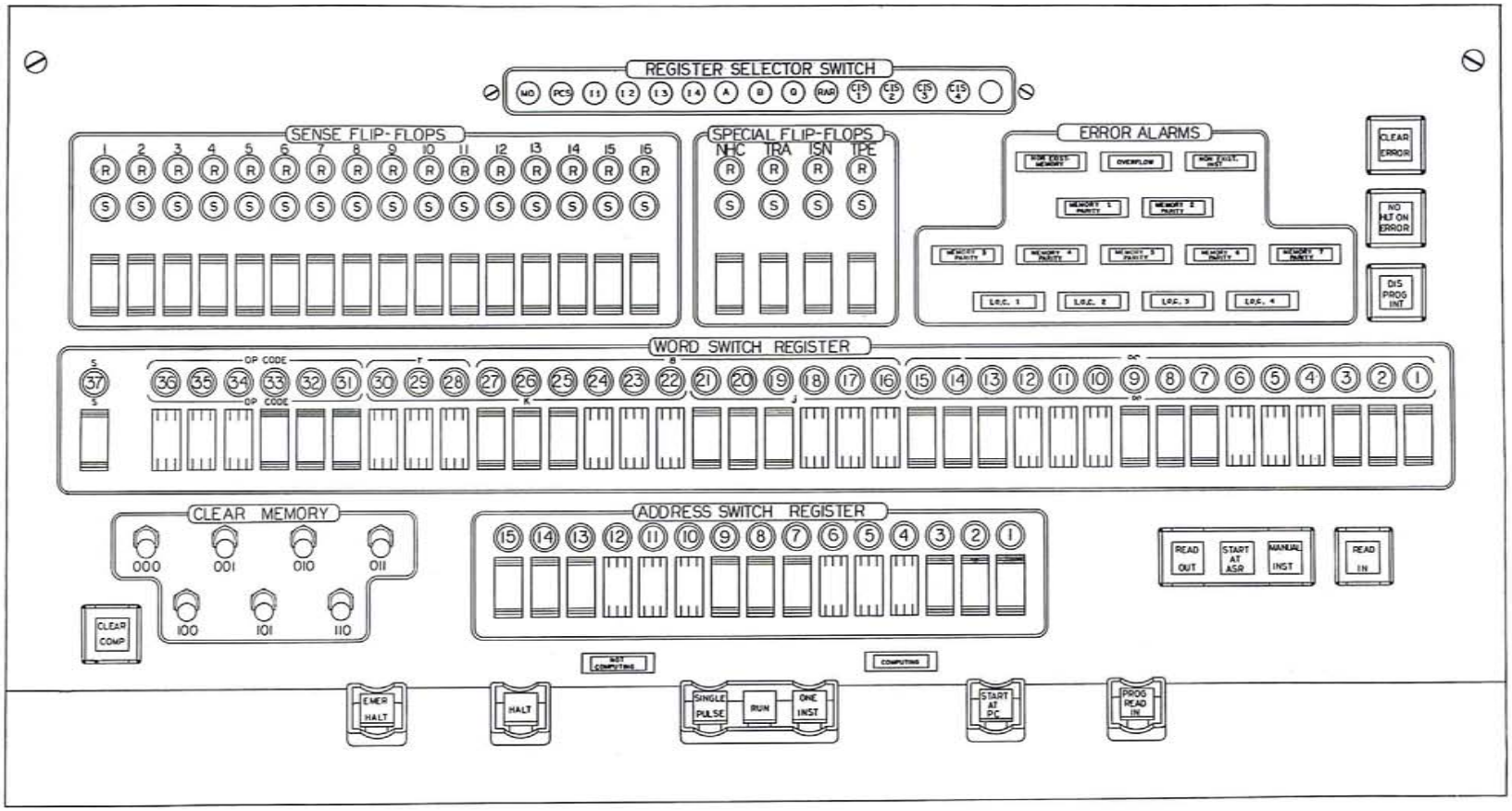


FIGURE 7-21 CONSOLE PANEL AND STEP PANEL

19. TABLE 4 - ERROR DETECTION

Type of Error	Addressable Flip-Flop	Effect	Comment
Non-Existent Memory	No	Stops Machine	Unless KEW switch on console is depressed.
Non-Existent Instruction	No	Stops Machine	Unless KEW switch on console is depressed.
Machine Parity	No	Stops Machine	Unless KEW switch on console is depressed. Machine parity is checked on read in and out of memory.
Overflow	0100	Determined by the bits (16, 17 and 18) of the order causing overflow.	If effect is to Stop, this will not happen if the KEW switch is depressed.
Input - Output Converter Alarms	0130 0131	If NHC flip-flop (0103) is set, it will not halt except for DVA. If KEW is set, it will not stop on Input-Output Errors.	Caused by - DVA - Device Alarm RPE - Input Parity ISN - Interpret Sign TRE - Timing IMP - Improper Order
Real Time Input Parity Error	0104	Sets RPE which can only be cleared by a SNR instruction from program.	Does not stop the computer

20. DIAGNOSTIC AND CHECKOUT PROCEDURES

20.1 The Types of Errors in Written Programs

There are three very common types of errors that can occur in written programs.

a) Closed loops

b) Incorrect answers

c) Unexpected halt before program reaches completion

The "finding errors" stage of programming is called "debugging".

20.2 Types of "Debugging" Routines

Five types of "debugging" routines which aid the programmer in finding his or her mistakes.

a) The Trace

1. Complete. Traces every instruction.

2. Selective. Traces every transfer.

3. Partial. Gives the programmer the option of tracing a certain part of his or her program.

b) Memory Dumps

1. Periodic. Shows what is in memory at a certain point of the program.

2. Single. Shows what is in memory at the end of a program.

c) Analyzers show the places in a program which refer to a particular location. The analyze routine, when used with a program, will produce a list of all reference to particular locations.

d) Use of sense switches to indicate paths.

e) Use of trapping mode.

20.3 Efficient Preparation of Programs

Efficient preparation of programs can be aided by using the following methods:

a) Small independent pieces of coding.

b) Use of prechecked routines or library subroutines wherever possible.

c) Hand checking of logic and coding.

d) Numerical handchecks.

e) Thorough annotation.

f) Use of sense switches.

21. TYPICAL EXAM

1. Add every seventh number from location 6000-8000 inclusive. Fill in the remaining parts of the instructions at locations 101, 102, & 103.

```
100  CLA  70000
101  LDX
102  ADD
103  TRX   ,    ,7
```

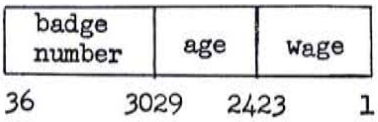
2. Assume bits 34, 35, 36, and 37 of the Q register contain ones, the rest of the accumulator and Q register contain zeros. After the following program, what bits in A & Q contain ones?

```
     SLL  36
     CYS  2
     CYL  1
     SLL  1
```

3. The following program is to extract β from location 200 and place it in the accumulator. Fill in the missing instruction in location 52.

```
50   1777000077777
51   CLA  50
52
53   LGM  200
```

4. An electronics firm retires its employees when they reach 65 years of age. The company keeps the records of each of its 200 employees according to badge number, in ascending order, on mag tape 1, in one block, the word formation being:



The information on mag tape is three years old. Count up the number of employees who have retired within that time and print each one's information word on paper tape. Also count the number of employees to be retired next year, using loc. 701 as a counter.

Answer to Problem #4

```
20)  RAN  1000,MT1,200      Put tape information in memory
21)  MOV  70000,700         Clear counters
22)  MOV  70000,701
23)  LDX  200,1,0           200 ---> I2, 0 ---> I1
24)  SEN  24,,MT1           Sense for completion of mag tape
25)  CLA  1000,1            First information word --> Acc
26)  LGM    50              Isolates the age
27)  SUB    52              If 61, (loc.52) retired next yr.;
                            more than 61, already retired
30)  TRZ    41              Transfer if retired next year
31)  TRP    43              Transfer if already retired
32)  TRX  25,1,1            Information word is still under
                            65, so go back and examine next
                            one
33)  CLA   700              Put counter for those retired
                            in Acc
34)  SHL    21              Shift into "k" position
35)  LOD  51,,0011          Load mask
36)  MSK  37                Mask into the write order, k words
37)  WOK  500,PTP,**        Write k words
40)  HLT
41)  ADB  701,0,1           Increase counter for those
                            retired next year
```

```
42)  TRU  32                    Go back and examine next information word

43)  CLA  1000,1                For those already retired, put into the whole information word

44)  STR  500                   into loc. 500 for writing or paper tape, then add one to the retired
45)  ADB  700,0,1               counter (700), and also to the storage loc. to prepare for next
46)  ADB   44,0,1               word

47)  TRU   32                   Go back and examine next number

50)  + 003740---------0
51)  + 000770---------0         Dummy orders
52)  |0----0|  61  |0---0|
       36   3029   24    1
```

22. MOBIDIC PROGRAMMER AIDS

22.1 Addressable Registers

Register Address (OCTAL)	Register Name	Code
00000 - 07777	Memory Unit Zero	
10000 - 17777	Memory Unit One	
20000 - 27777	Memory Unit Two	
.	.	
.	.	
.	.	
60000 - 67777	Memory Unit Six	
70001	Index Register One	(I^1)
70002	Index Register Two	(I^2)
70003	Index Register Three	(I^3)
70004	Index Register Four	(I^4)
70010	Accumulator	A
70011	Q Register	Q
70012	B Register	B
70013	Program Counter	PC
70014	Program Counter Store	PCS
70015	Instruction Register of Converter receiving the input order	CRI
70020	Word Switch Register	WSR
70021	Real Time Address Register	RAR
70022	Real Time Output Register	ROR
70030	Instruction Register of first in-out converter	CIS1
70031	Instruction Register of second converter	CIS2

22.2 MOBIDIC OPERATIONAL CODES LISTED IN ALPHABETICAL ORDER WITH PROGRAMMING MANUAL PAGE REFERENCES

Oper. Abbrev.	Octal Code	Address Fields Used	μ Sec	Operation	Page	Indexable
ADB	24	$\gamma\beta\alpha$	26	Add Beta	29	No
ADD	12	$\gamma\beta\alpha$	16	Add	20	Yes
ADM	13	$\gamma\beta\alpha$	16	Add Magnitude	21	Yes
BSP	67	$\kappa\tau$	16 + 144 per word	Backspace	37	No
CAM	11	$\gamma\alpha$	16	Clear & Add Magnitude	22	Yes
CLA	10	$\gamma\alpha$	16	Clear & Add	17	Yes
CLS	14	$\gamma\alpha$	16	Clear & Subtract	22	Yes
CSM	15	$\gamma\alpha$	16	Clear & Subtract Mag.	22	Yes
CYL	35	$\gamma\alpha$	$\alpha \leq 14 = 16$	Cycle Long	38	Yes
CYS	34	$\gamma\alpha$	$\alpha > 14, 2 + \alpha + \alpha$ (MOD2)	Cycle Short	38	Yes
DVD	22	$\gamma\beta\alpha$	88 (18 if overflow)	Divide	23	Yes
DVL	23	$\gamma\beta\alpha$	88 (18 if overflow)	Divide Long	23	Yes
HLT	00		16	Halt	27	No
LDX	53	$\gamma\beta\alpha$	16	Load Index	29	No
LGA	03	$\gamma\alpha$	16	Logical Add	38	Yes
LGM	02	$\gamma\alpha$	16	Logical Multiply	38	Yes
LGN	04	$\gamma\alpha$	16	Logical Negation	39	Yes
LOD	51	$\gamma\beta\alpha$	18	Load	17	Yes

Oper. Abbrev.	Octal Code	Address Fields Used	μ Sec	Operation	Page	Indexable
MLR	21	γα	86	Multiply & Round	22	Yes
MLY	20	γα	86	Multiply	22	Yes
MOV	52	γβα	26	Move	17	No
MSK	55	γα	22	Replace Through Mask	17	Yes
NRM	37	γα	18 + m - m(mod2)	Normalize	23	Yes
RAN	70	K,J,α	16 + 8 per word	Real Alphanumeric	36	No
ROK	72	K,J,α	16 + 8 per word	Read Octal	35	No
RPA	54	γα	22	Replace Address	17	Yes
RPT	01	γβα	16	Repeat	30	Yes
RRV	71	K,J,α	16 + 144 per word	Read Reverse	37	No
RWD	77	J	16 + 144 per word	Rewind	37	No
SBB	25	γβα	25	Subtract Beta	29	No
SBM	17	γβα	16	Subtract Magnitude	21	Yes
SEN	05	γβα	16	Sense	27	Yes
SHL	30	γβα	16 or more	Shift Left	24	Yes
SHR	32	γβα	16 or more	Shift Right	23	Yes
SKP	66	KJ	16 + 144 per word	Skip	37	No
SLL	31	γβα	16 or more	Shift Left Long	24	Yes
SNR	07	γβα	16	Sense & Reset	27	Yes
SNS	06	γβα	16	Sense & Set	27	Yes
SRL	33	γα	16 or more	Shift Right Long	23	Yes
STR	50	γα	16	Store	17	Yes
SUB	16	γβα	16	Subtract	20	Yes

Oper. Abbrev.	Octal Code	Address Fields Used	Sec	Operation	Page	Indexable
TRC	47	$\gamma\alpha$	22	Compare	27	Yes
TRL	41	$\gamma\beta\alpha$	16	Load P.C.S. & Trans.	29	No
TRN	46	$\gamma\alpha$	16	Trans. on Negative Acc.	26	Yes
TRP	44	$\gamma\alpha$	16	Trans. on Positive Acc.	26	Yes
TRS	42		16	Trans. to Program Cnt. Store	30	No
TRU	40	$\gamma\beta\alpha$	16	Uncond. Transfer	25	Yes
TRX	43	$\gamma\alpha$	22	Trans. on Index	29	No
TRZ	45	$\gamma\alpha$	16	Trans. on Zero Acc.	27	Yes
WAN	74	$\kappa\jmath\alpha$	16 + 8 per word	Write Alphanumeric	34	No
WOK	76	$\kappa\jmath\alpha$	16 + 8 per word	Write Octal	35	No
WWA	75	$\kappa\jmath\alpha$	16 + 144 per word	Rewrite	37	No

22.3

MOBIDIC OPERATIONAL CODES LISTED IN NUMERICAL CODING ORDER WITH PROGRAMMING MANUAL PAGE REFERENCES

OCTAL CODE	OPER. ABBREV.	ADDRESS FIELDS USED	μ SEC	OPERATION	PAGE	INDEX-ABLE?
00	HLT		16	Halt	27	No
01	RPT	$\gamma\beta\alpha$	16	Repeat	30	Yes
02	LGM	$\gamma\alpha$	16	Logical Multiply	38	Yes
03	LGA	$\gamma\alpha$	16	Logical Add	38	Yes
04	LGN	$\gamma\alpha$	16	Logical Negation	39	Yes
05	SEN	$\gamma\beta\alpha$	16	Sense	27	Yes
06	SNS	$\gamma\beta\alpha$	16	Sense & Set	27	Yes
07	SNR	$\gamma\beta\alpha$	16	Sense & Reset	27	Yes
10	CLA	$\gamma\alpha$	16	Clear & Add	17	Yes
11	CAM	$\gamma\alpha$	16	Clear & Add Magnitude	22	Yes
12	ADD	$\gamma\beta\alpha$	16	Add	20	Yes
13	ADM	$\gamma\beta\alpha$	16	Add Magnitude	21	Yes
14	CLS	$\gamma\alpha$	16	Clear & Subtract	22	Yes
15	CSM	$\gamma\alpha$	16	Clear & Sub-tract Mag.	22	Yes
16	SUB	$\gamma\beta\alpha$	16	Subtract	20	Yes
17	SBM	$\gamma\beta\alpha$	16	Subtract Magnitude	21	Yes
20	MLY	$\gamma\alpha$	86	Multiply	22	Yes
21	MLR	$\gamma\alpha$	86	Multiply & Round	22	Yes
22	DVD	$\gamma\beta\alpha$	88 (18 if over flow)	Divide	23	Yes
23	DVL	$\gamma\beta\alpha$	88 (18 if over flow)	Divide Long	23	Yes
24	ADB	$\gamma\beta\alpha$	26	Add Beta	29	No

TAL CODE	OPER. ABBREV.	ADDRESS FIELDS USED	SEC	OPERATION	PAGE	INDEX-ABLE?
25	SBB	$\gamma\beta\alpha$	25	Subtract Beta	29	No
26						
27						
30	SHL	$\gamma\beta\alpha$	$\begin{cases}\alpha \le 9=16;\\ \alpha > 9,8+\\ \alpha-\alpha\ (\mathrm{mod}2)\end{cases}$	Shift Left	24	Yes
31	SLL	$\gamma\beta\alpha$		Shift Left Long	24	Yes
32	SHR	$\gamma\alpha$	$\begin{cases}\alpha \le 14=16;\\ \alpha > 14,2+\\ \alpha-\alpha\ (\mathrm{mod}2)\end{cases}$	Shift Right	23	Yes
33	SRL	$\gamma\alpha$		Shift Right Long	23	Yes
34	CYS	$\gamma\alpha$	$\begin{cases}\alpha \le 14=16\\ \alpha > 14,2+\\ \alpha+\alpha\ (\mathrm{mod}2)\end{cases}$	Cycle Short	38	Yes
35	CYL	$\gamma\alpha$		Cycle Long	38	Yes
37	NRM	$\gamma\alpha$	$\begin{cases}18+n-n\ (\mathrm{mod}2)\\ n=\#\ \text{of shifts}\end{cases}$	Normalize	23	Yes
40	TRU	$\gamma\beta\alpha$	16	Uncond. Transfer	25	Yes
41	TRL	$\gamma\beta\alpha$	16	Load P.C.S. & Trans.	29	No
42	TRS		16	Trans. to Program Cnt. Store	30	No
43	TRX	$\gamma\beta\alpha$	22	Trans. on Index	29	No
44	TRP	$\gamma\alpha$	16	Trans. on Positive Acc.	26	Yes
45	TRZ	$\gamma\alpha$	16	Trans. on Zero Acc	27	Yes
46	TRN	$\gamma\alpha$	16	Trans. on Negative Acc.	26	Yes
47	TRC	$\gamma\alpha$	22	Compare	27	Yes
50	STR	$\gamma\alpha$	16	Store	17	Yes
51	LOD	$\gamma\beta\alpha$	18	Load	17	Yes
52	MOV	$\gamma\beta\alpha$	26	Move	17	No
53	LDX	$\gamma\beta\alpha$	16	Load Index	29	No
54	RPA	$\gamma\alpha$	22	Replace Address	17	Yes
55	MSK	$\gamma\alpha$	22	Replace Through Mask	17	Yes

OCTAL CODE	OPER. ABBREV.	ADDRESS FIELDS USED	SEC.	OPERATION	PAGE	INDEX-ABLE?
56						
57						
58						
59						
60						
61						
62						
63						
64						
65						
66	SKP	KJ	16+144 per word	Skip	37	No
67	BSP	KJ	16+144 per word	Backspace	37	No
70	RAN	KJα	16+8 per word	Read Alpha-numeric	36	No
71	RRV	KJα	16+144 per word	Read Reverse	37	No
72	ROK	KJα	16+8 per word	Read Octal	35	No
73						
74	WAN	KJα	16+8 per word	Write Alpha-numeric	34	No
75	WWA	KJα	16+144 per word	Rewrite	37	No
76	WOK	KJα	16+8 per word	Write Octal	35	No
77	RWD	J	16+144 per word	Rewind	37	No