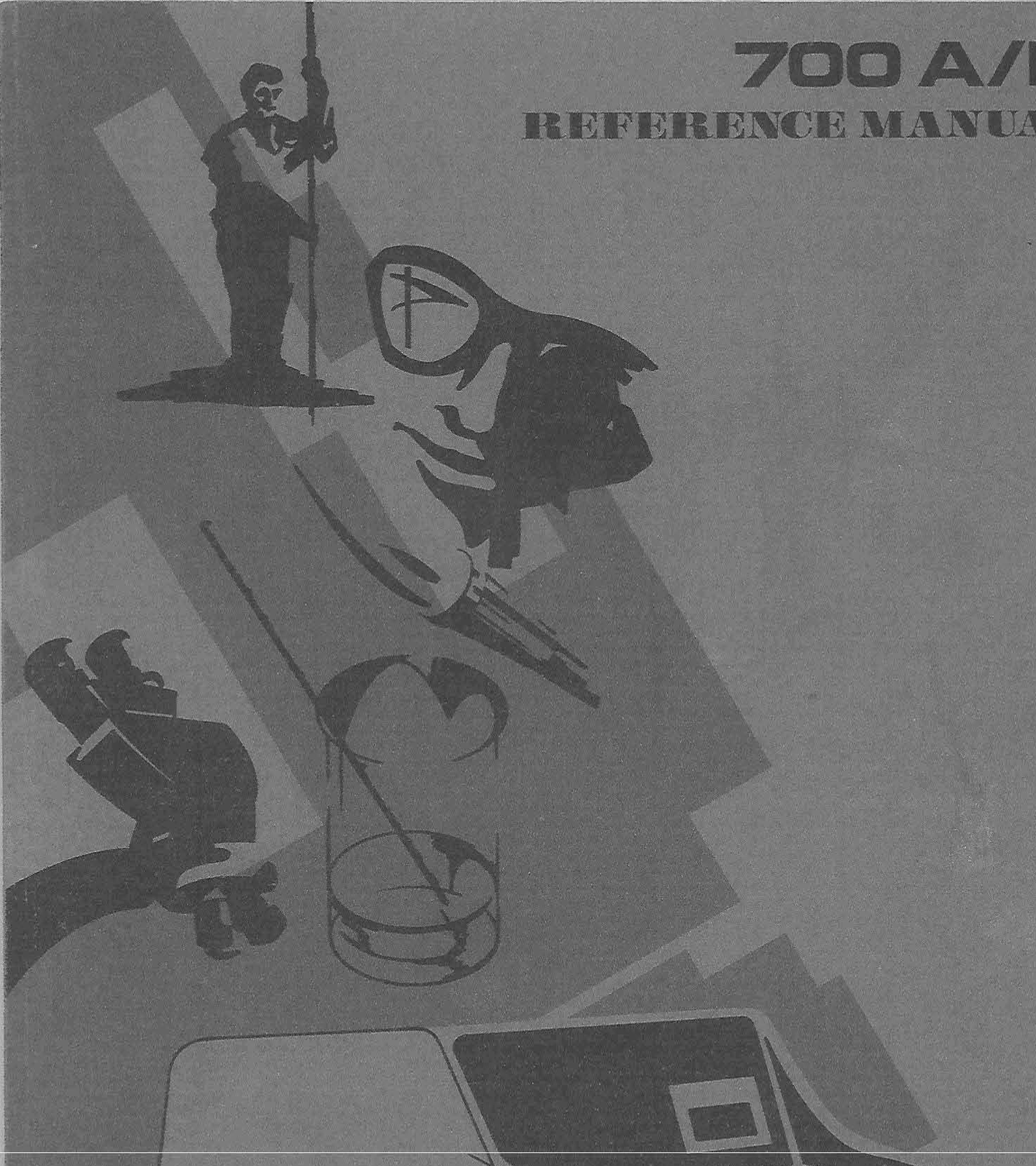
700 A/B

REFERENCE MANUAL



WANG
LABORATORIES, INC.

700 A/B

REFERENCE MANUAL

© WANG LABORATORIES, INC., 1970
Tewksbury, Mass. 01876
Telephone (617) 851-7311
TWX 710 343-6769

FOREWORD

This reference manual is designed to provide the user with a basic understanding and practical guidance in the use of Wang's 700A/B Electronic Calculators.

The aim has been to assist the user by presenting the most useful technique, concept and method for utilizing the 700 to its best advantage.

For further information, contact your local sales office or Wang Laboratories, Inc., 836 North Street, Tewksbury, Massachusetts 01876.

TABLE OF CONTENTS

SECTION I – INTRODUCTION

SECTION II – EXPLANATION OF KEYS

SECTION III – PROGRAMMING

SECTION IV – PROGRAM CONCEPTS

TABLE OF CONTENTS (Continued)

SECTION V – DECISION COMMANDS

SECTION VI – PROGRAMMING TECHNIQUES USING A TAPE CASSETTE

SECTION VII – ADDITIONAL COMMANDS NOT FOUND ON
THE 700 KEYBOARD

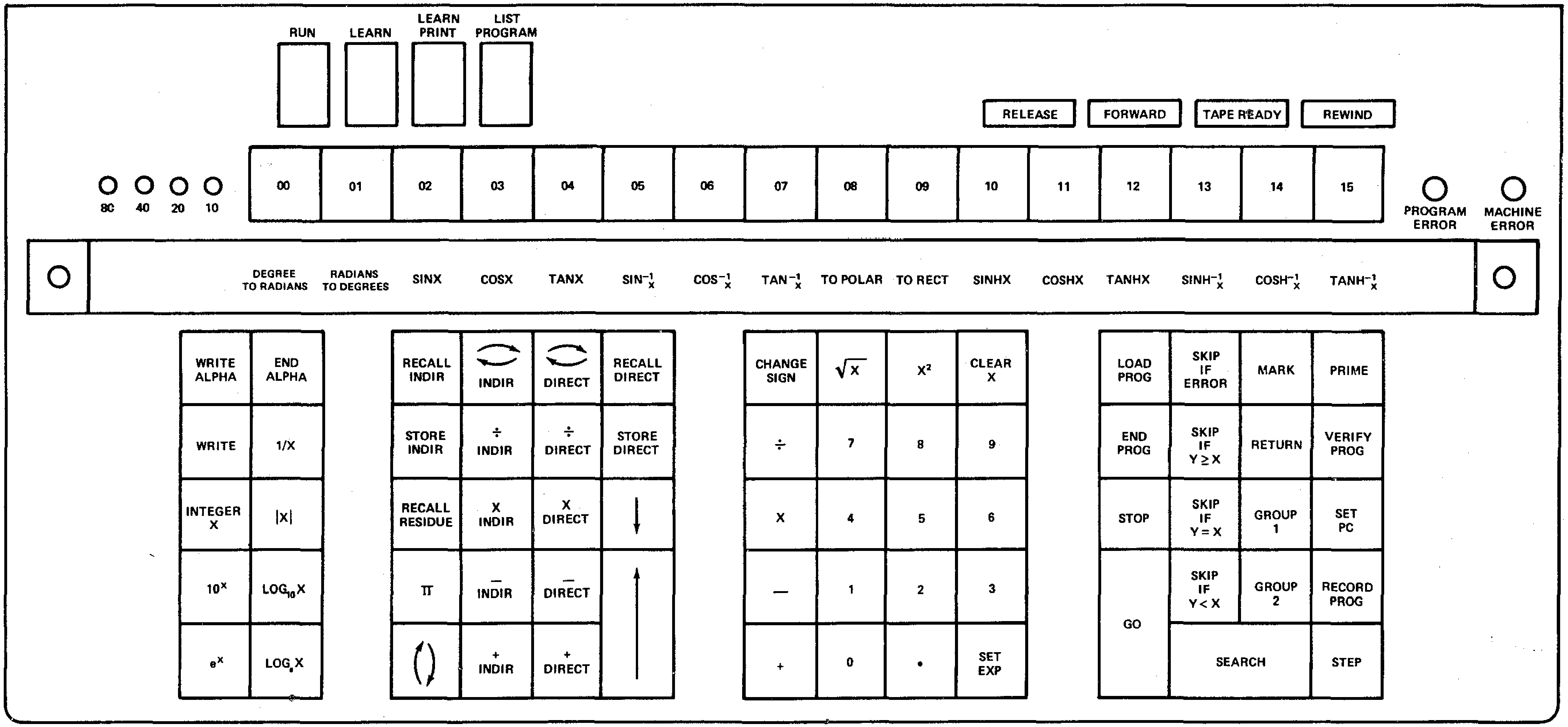
SECTION VIII – TRIGONOMETRIC PACKAGE PROGRAM
STATISTICAL PACKAGE PROGRAM

TABLE OF CONTENTS (Continued)

SECTION IX – SAMPLE PROGRAMS

SECTION X – WARRANTY, SERVICE AND MAINTENANCE

APPENDIX



700A/B KEYBOARD ILLUSTRATION

SECTION I
INTRODUCTION

The Wang 700 is the ultimate solution to many calculating needs. Simple or complex calculations can be done right at the desk. The 700 keyboard is extremely simple to operate. Once the fundamental operations have been mastered, programming the 700 is easy.

The Wang 700 is a self-contained programmable electronic calculator constructed with integrated circuits on snap-in replaceable printed circuit modules. The 700 is composed of three basic elements:

1. The Central Processing Unit
2. Read-Only Memory
3. Core Memory

The Central Processing Unit (CPU) is the hardware which performs the arithmetic operations. The **Read-Only Memory guides** the CPU in all its operations. In effect, the Read Only Memory is "the brains" of the Wang 700. It directs all arithmetic and logical operations on the 700 and has been programmed to perform all the functions found on the 67 keys of the 700 keyboard.

The 700 Core Memory is organized into 121 or 122 data registers; plus a nixie display of the two work registers X and Y; 120 registers can be used for data storage or program storage. All user programs are executed from core memory. The trig functions are also executed from core memory.

Section II explains the modes of operation on the 700, as well as the five non-programmable keys; discusses the dual nixie-type display readout and the basic arithmetic operations; explains direct and indirect addressing of the data storage registers; explains the **RECALL RESIDUE** key – a unique Wang feature that makes double-precision arithmetic a simple operation on the 700.

The usefulness of the Wang 700 comes from its programming capabilities. A program is simply a logical sequence of steps which the calculator can perform automatically over and over again on different variables. If the calculation is to be performed only once, it probably is simple enough to do it manually on the keyboard. However, if the same calculations are to be done repeatedly, it is beneficial to record and save the steps of the calculation in the form of a program and let the calculator perform these repeated operations. The program is loaded into core and executed from core. Programs can also be stored on magnetic tape for later use.

Sections III, IV, V, VI, and VII discuss various techniques to use in programming the Wang 700. They explain how to introduce a program into core memory and how to save it for later use on tape cassettes. Also, they explain how several parts of a program can share the same part of core memory. Section VIII discusses the TRIG functions of the Wang

700. Section IX gives an example of a 700 Program and further illustrates the concept of indirect addressing. Section X contains warranty, service, and maintenance information.

An appendix is included in this manual which covers typing conventions and contains an index to help the user locate with ease certain items of interest.

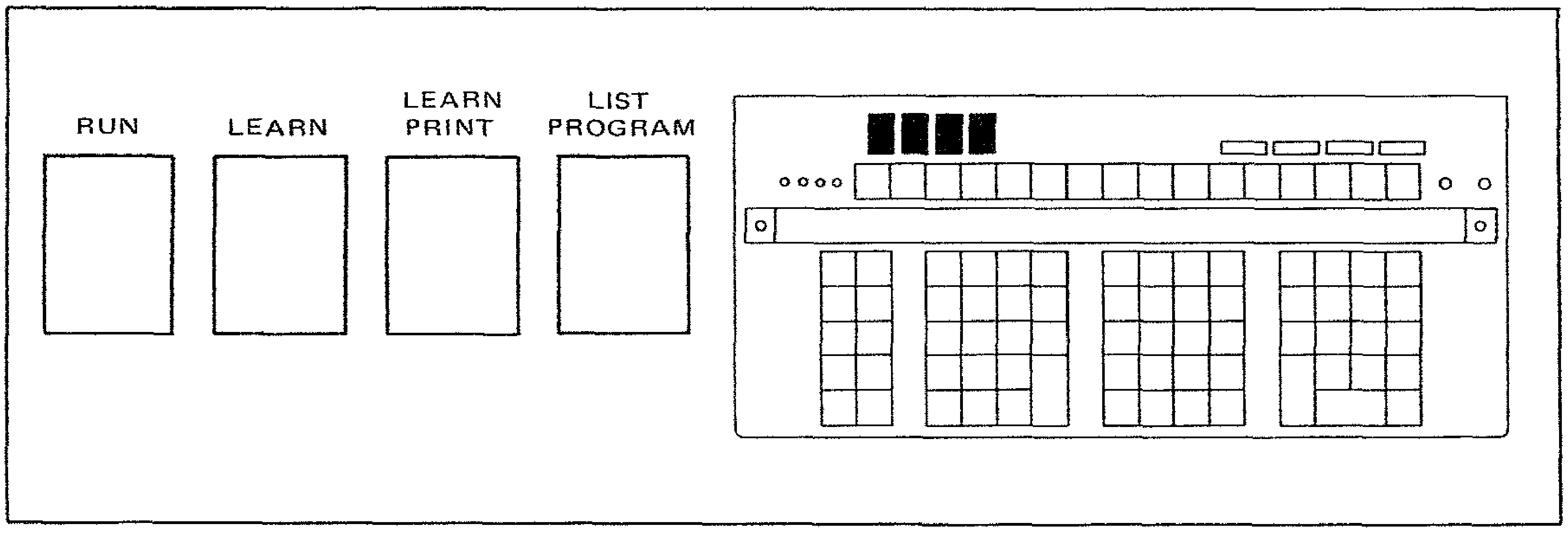
700A – 700B

	PROGRAM	
	STEPS	REGISTERS
700A	960	000–119 120–121* (Scratch Pad Only) 2 Level subroutine Drives 701 output writer
700B	960	000–119 120* (Storage Only) 5 Level subroutine Drives 701 Output writer 702 Plotter

*The 700A register 121 and the 700B register 120 may be used as scratch pads only if subroutine 00-00 thru 01-15 are not accessed. When these subroutines are called upon the Y register is automatically stored in these registers.

SECTION II
EXPLANATION OF KEYS

MODES OF OPERATION



The 700 has four different modes of operation. The four lock-in switches located above the toggle switches on the 700 keyboard are used to put the 700 into a certain mode of operation.

RUN MODE

The RUN MODE is used for most 700 operations. All keyboard calculations are done in the RUN MODE. In fact, practically all operations except introducing a program into core memory from the keyboard are performed on the 700 in the RUN MODE.

LEARN MODE

The 700 is put into LEARN MODE when a program is to be written into core. Every programmable key which is indexed while the 700 is in LEARN MODE is "learned" or recorded in core. In LEARN MODE the Y-Register is blanked and the X-Register displays the program step number and the program code stored at that step.

LEARN-PRINT MODE

The LEARN-PRINT MODE and the LIST PROGRAM MODE are used only when the output writer is available. In the LEARN-PRINT MODE,each key indexed is "learned" into core and is also listed on the output writer. As each key is indexed, the program step number and the program code of the key is listed on the output writer, giving the user a hard copy of his program as he writes it!

LIST-PROGRAM MODE

When the 700 is put in the LIST-PROGRAM MODE and the **GO** key is depressed, it automatically lists the program steps and program code in increments of 100 steps until it encounters an **END PROGRAM** code. The LEARN-PRINT and LIST-PROGRAM modes are discussed in greater detail in the 701 OUTPUT WRITER MANUAL.

TURNING THE 700 ON

The procedure for turning the 700 on consists of three steps:

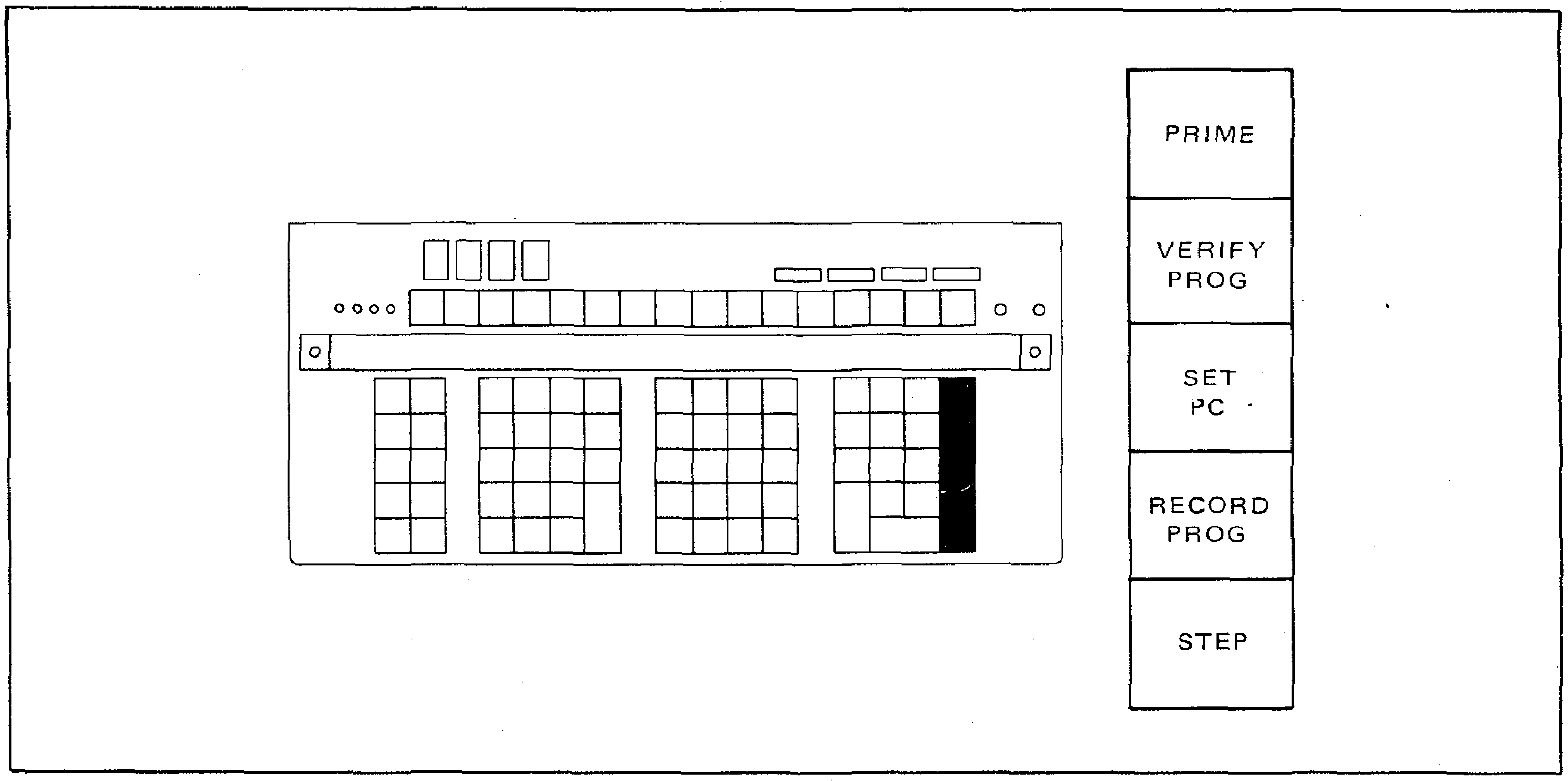
1. *Turn power switch ON.*
2. *Index PRIME to initialize the system.*
3. *Select mode of operation. (In most instances the RUN mode will be selected. Depress RUN button.)*

The Wang 700 is now in RUN MODE ready to perform your calculations.

NON PROGRAMMABLE KEYS

Because of their function, there are five keys which cannot be programmed on the Wang 700. Each of these commands is discussed briefly in this section. All of their functions and uses will become clear after reading the entire manual.

The five non programmable keys on the 700 are:



PRIME

The **PRIME** key initializes the 700 system and should always be depressed when the 700 is first turned on. It also performs the following operations:

1. Clears Y-Register to zero.
2. Clears X-Register to zero.
3. Sets the program counter to Step 000.
4. Resets program-error and machine-error indicators.

The PRIME key should be depressed when the 700 is first turned on.

NOTE

The **PRIME** *key should not be depressed when any operation is being executed. If the program is to be stopped during execution, the* **STEP** *key should be used. This will stop the program after the current step is executed. Also indexing the* **PRIME** *key when* **RECORD PROGRAM** *or* **LOAD PROGRAM** *commands are being executed will cause difficulty with the tape. If the* **PRIME** *key is indexed accidentally during a* **RECORD PROGRAM** *or* **LOAD PROGRAM** *operation, the operation will be terminated immediately. However, the tape should be rewound before executing any other tape operations.*

PROGRAM COUNTER AND SET PC

The program counter or PC is a counter which counts from 000 to 959. It indicates which program step is about to be executed. At all times, it always points to the **next** program step. Thus, when the machine is performing step 108, the PC is already on step 109.

The **SET PC** key allows the user to address and set the program counter with the next three keystrokes.

SET PC 0 1 8

This instruction sets the program counter to program step number 018. To set the program counter requires four keystrokes: **SET PC** followed by three numeric keys. **PRIME** automatically sets the program counter to step number 000.

STEP

The **STEP** key allows the user to step through his program one step at a time. If the program is running when the **STEP** key is indexed, the program stops at the step it is about to execute. In the RUN MODE, depressing the **STEP** key will cause the 700 to perform the next step in the program. Each time the **STEP** key is indexed, the next program step will be executed.

The **GO** key will take the 700 out of the stepping mode and put it in the continuous mode executing the remaining steps in the program until a **STOP** command is encountered.

NOTE

In any 2-step command such as **DIRECT ADDRESSING** *and* **WRITE ALPHA** *commands, the* **GO** *key should not be depressed in the middle of the 2-step command. The entire 2-step command should be executed in step mode before switching to the continuous mode.*

This stepping feature is of tremendous value for debugging programs. The programmer can step through his program and locate his difficulty immediately. By switching to LEARN

MODE he can see the step number and the code of the operation he is **about to execute.** When stepping through a program in LEARN MODE, the program step number (the PC) and the program code of the operation is displayed in the X-Register. However, in **LEARN MODE** the operation is not executed.

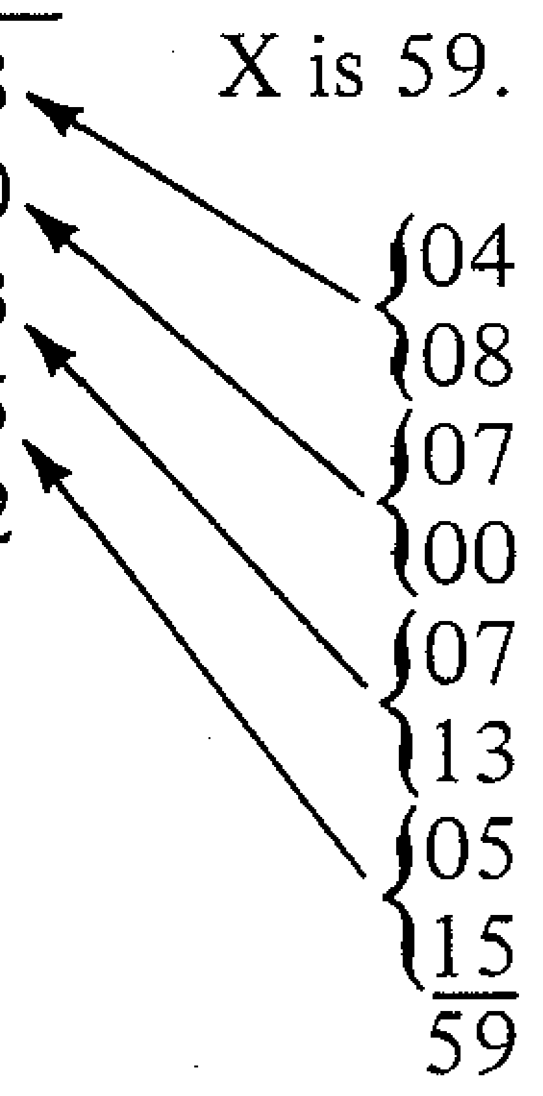
VERIFY PROGRAM

The **VERIFY PROGRAM** key decimally adds the high-order and low-order digits of the program codes in core beginning at step 000 until it encounters an **END PROGRAM** code. The sum is displayed in the right-most digits of the mantissa of the X-Register.

EXAMPLE

If this program is located in core and the VERIFY PROGRAM key is depressed the sum displayed in X is 59.

STEP #	KEY	CODE
000	MARK	0408
001	0700	0700
002	x^2	0713
003	STOP	0515
004	END PROG	0512



After performing this operation, the PC is set at the step where the **END PROGRAM** command is located. (Step 004 in this example.) Notice the code for **END PROGRAM** is not added into the sum generated by the **VERIFY PROGRAM** key.

RECORD PROGRAM

This key transfers a program from core to magnetic tape. The PC is set to a specific step and the program steps starting at this step are transferred to the tape until an **END PROGRAM** command is reached. The **END PROGRAM** command is the last step transferred to the tape. After transferring the program to tape, the PC is set to where it was originally set (i.e., the first program step to be transferred to the tape).

The five keys, **PRIME, SET PC, VERIFY PROGRAM, RECORD PROGRAM** and **STEP** are the only keys on the 700 which cannot be used in a program. Therefore, when any of these keys are indexed it doesn't matter whether the machine is in LEARN or RUN MODE.

THE DISPLAY

The display consists of two work registers, X and Y. Both the X and Y Registers are displayed simultaneously by easily readable half-inch nixie-type tubes. Each register has a ± sign and twelve digit mantissa followed by a two-digit exponent with a range of –99 to +99.

```
± · X X X X X X X X X X X X    ± X X        (Y-Register)
± · X X X X X X X X X X X X    ± X X        (X-Register)
\  \         mantissa           \  \exponent
 \  floating decimal          sign of exponent
sign of mantissa
```

For numbers in the range .1 ⩽ |N| < 1 000 000 000, the decimal point retains its natural position. When a number lies outside this range, the decimal automatically relocates to the extreme left, and the exponent of the power of 10 is indicated correctly in modified scientific notation. This property will become clear after a few minutes familiarization with the keyboard.

(A few numbers and how they appear in the display are given below.)

X-REGISTER

The keys **0**, **1**, 2, . . .9 and decimal point (.) are used for entering a number into the X-Register. The **SETEXP** key is used to set the exponent value of X. The **CH SIGN** key changes the algebraic sign of the mantissa or exponent of X.

Indexing a number into the 700 keyboard is as simple as writing the number down on paper. The normal sequence of steps is to key in the mantissa followed by the **SETEXP** key and the value of the exponent.

ENTERING A NUMBER

Index the following few numbers on the 700 keyboard. After indexing the number into the X-Register, move it to the Y-Register by depressing the ↑ key.

NUMBER	SEQUENCE OF STEPS	DISPLAY
a) $.152 \times 10^{21}$	1 5 2 SETEXP 2 1 ↑	.152000000000+21
b) 6.62517×10^{-27}	6 6 2 5 1 7 SETEXP CHS 2 6 ↑	.662517000000−26
c) −2534.5	2 5 3 4 . 5 CHS ↑ Or 2 5 3 4 5 CHS SETEXP 4 ↑	−2534.50000000
d) .0075	. 0 0 7 5 ↑ Or 7 5 SETEXP CHS 2 ↑	+.750000000000−02

Index c and d both ways. Does the display appear differently? Notice example b. Why is 26 entered as the value of the exponent?

SET EXP

The **SETEXP** key is used to set the exponent value of X with the next two successive keystrokes. The **SETEXP** key automatically aligns the decimal point in the left-most position of the X-Register: however, it does not blank out the mantissa. This allows us to change the value of the exponent of a number without having to key in the entire number again.

EXAMPLE

Index 1.75 x 10^{23}
Suppose the following sequence of steps is used:

1 . 7 5 SETEXP 2 3

Notice what happens to the decimal point when the **SETEXP** key is indexed. It is not necessary to index the decimal point, as the **SETEXP** key automatically aligns it in the left most position. The value of the exponent will also have to be indexed correctly. If the number is in proper scientific notation, the value of the exponent is simply increased by 1. Thus, the correct sequence of steps would be:

CLEAR X 1 7 5 SETEXP 2 4

All numbers indexed after the **SETEXP** key simply changes the value of the exponent. Since the range of the exponent is –99 to +99, normally only 1 or 2 numbers are indexed after the **SETEXP** key. However, if more than 2 numbers are indexed, the exponent takes on the value of the last 2 numbers entered.

EXAMPLE

If the following sequence of steps is performed:

1. 1 2 **SETEXP** 2 3 4, the value of the exponent is 34.
2. For **SETEXP CHS** 3 5 7, the value of the exponent is –57.
3. For **SETEXP** 5 0 2, the value of the exponent should be 2. However, on the display the exponent would be blanked out and the decimal point would assume its natural position.

The 700 will remain in the SETEXP mode until a non numeric key or the decimal point key is depressed.

Y-REGISTER

The Y-Register is another work register used in conjunction with the X-Register for basic arithmetic operations and data transfers. A number in the X-Register can easily be transferred to the Y-Register by indexing the ↑ key or ↓↑ key.

KEYSTROKE	OPERATION
CLEAR X	Clears X-Register
↑	X into Y, X unchanged
↓	Y into X, Y unchanged
↓↑	X and Y exchanged
+	Y+X into Y, X unchanged
–	Y–X into Y, X unchanged
X	Y×X into Y, X unchanged

÷	Y÷X into Y, X unchanged
|X|	Absolute value of X into X, Y unchanged
INT X	Disregards decimal part of number in X and puts integer part of number in X, Y unchanged
1/X	1/X into X, Y unchanged
x^2	x^2 into X, Y unchanged
$\sqrt{x}$	$\sqrt{x}$ into X, Y unchanged
$LOG_{10}X$	$LOG_{10}X$ into X, Y unchanged
10^X	10^X into X, Y unchanged
LOG_eX	LOG_eX into X, Y unchanged
e^X	e^X into X, Y unchanged
π	π into X, Y unchanged

Step through the following examples to familiarize yourself with these keyboard operations.

EXAMPLE 1. Calculate .083 + 17.86 + 32.2 = +50.1430000000

1. **PRIME**
2. . **0 8 3 ↑**
3. **1 7** . **8 6** +
4. **3 2** . **2** + (Answer in Y, 32.2 in X)

EXAMPLE 2. Calculate $(5)^2 - (20)^2 + (1/15)^2 + \sqrt{70} = -366.628955291$

1. **CLEAR X**
2. **5** x^2 ↑
3. **20** x^2 –
4. **1 5 1/x** x^2 +
5. **70** $\sqrt{x}$ + (Answer in Y, $\sqrt{70}$ in X)

EXAMPLE 3. Calculate $\frac{51}{\sqrt{26}}$ x 6.2 = +62.0119219307

1. **5 1** ↑	1. **5 1** ↑
2. **6** . **2** x	2. **2 6** $\sqrt{x}$ ÷
3. **2 6** $\sqrt{x}$ ÷	3. **6** . **2** x
(Answer in Y, $\sqrt{26}$ in X)	(Answer in Y, 6.2 in X)

EXAMPLE 4. $A = \pi r^2$ $r = .568 \times 10^{-6}$ = +.101355318827 - 11

1. **5 6 8 SETEXP 6 CHS**
2. x^2 ↑ π
3. X (Answer in Y, π in X)

EXAMPLE 5. Calculate $(12.8)^{7/3}$ = +383.256852976

1. 1 2 . 8 **LOG$_e$X**		1. 1 2 . 8 **LOG$_{10}$X**
2. ↑ 7 x	or	2. ↑ 7 x
3. 3 ÷		3. 3 ÷
4. ↓ **e^X** (Answer in X)		4. ↓ **10^X** (Answer in X)

EXAMPLE 6. Reduce the angle 865° to an equivalent angle less than 360°.

Formula $\left(\frac{865}{360} - \text{INT}\left[\frac{865}{360}\right]\right)360$ = equivalent (145) value

1. 8 6 5 ↑
2. 3 6 0 ÷
3. ↓ **INT (X)**
4. –
5. 3 6 0 x (Answer in Y, 360 in X)

EXAMPLE 7. Calculate the following:

a. $c = 2\pi r$ where $r = .347 \times 10^{-5}$ = .21802 . . x10^{-4}

b. $M = \frac{90 + 87 + 68 + 77}{4}$ = 80.5

c. $\sqrt{M} + \frac{1}{\sqrt{M}}$ where M is the answer of 7 (b)

Answer = 8.984 . . .

Hint: Use ↓↑ key

d. $e^{5.3} + 10^{5.7} + \pi^2$ = 501397 . 4 . . .

e. Log (–2) What happens? Why? **PRIME** and find $\sqrt{-3}$. What happens? Why?

PROGRAM-ERROR INDICATOR

There are two lights located to the right of the Special Function Keys on the 700 keyboard. These two lights are used as error indicators. The one on the right indicates MACHINE ERROR; the one on the left PROGRAM ERROR. The MACHINE ERROR INDICATOR is discussed later.

The PROGRAM ERROR INDICATOR is turned on whenever an illegal operation is performed (i.e., taking the logarithm or square root of a negative number, or dividing by zero). Also, if a calculated result is greater than 10^{99}, the PROGRAM ERROR INDICATOR will be turned on. Whenever the indicator is on, the arithmetic sign of the X-Register also flashes.

OPERATIONS WHICH TURN PROGRAM ERROR INDICATOR ON

Calculated result greater than 10^{99} (Overflow condition)
Division by 0
$\sqrt{-x}$
LOG$_{10}$x where $x \leqslant 0$

Log e^x where $x \leqslant 0$
Searching Non-Existent Flag (See page 4-3).
Addressing An Illegal Data Register (Any Register Greater than 121)
Program Overlaps Core (See BYPASSING PROGRAM BLOCKS page 6-8)
Program Block is Missing An **END PROGRAM** Instruction (See
Definition of PROGRAM BLOCK(page 6-4)

The **PRIME** key is used to turn the PROGRAM ERROR INDICATOR off. In programming, a **SKIP IF ERROR** command is available to test for this condition. Performing this test will also turn the indicator off.

DATA STORAGE REGISTERS

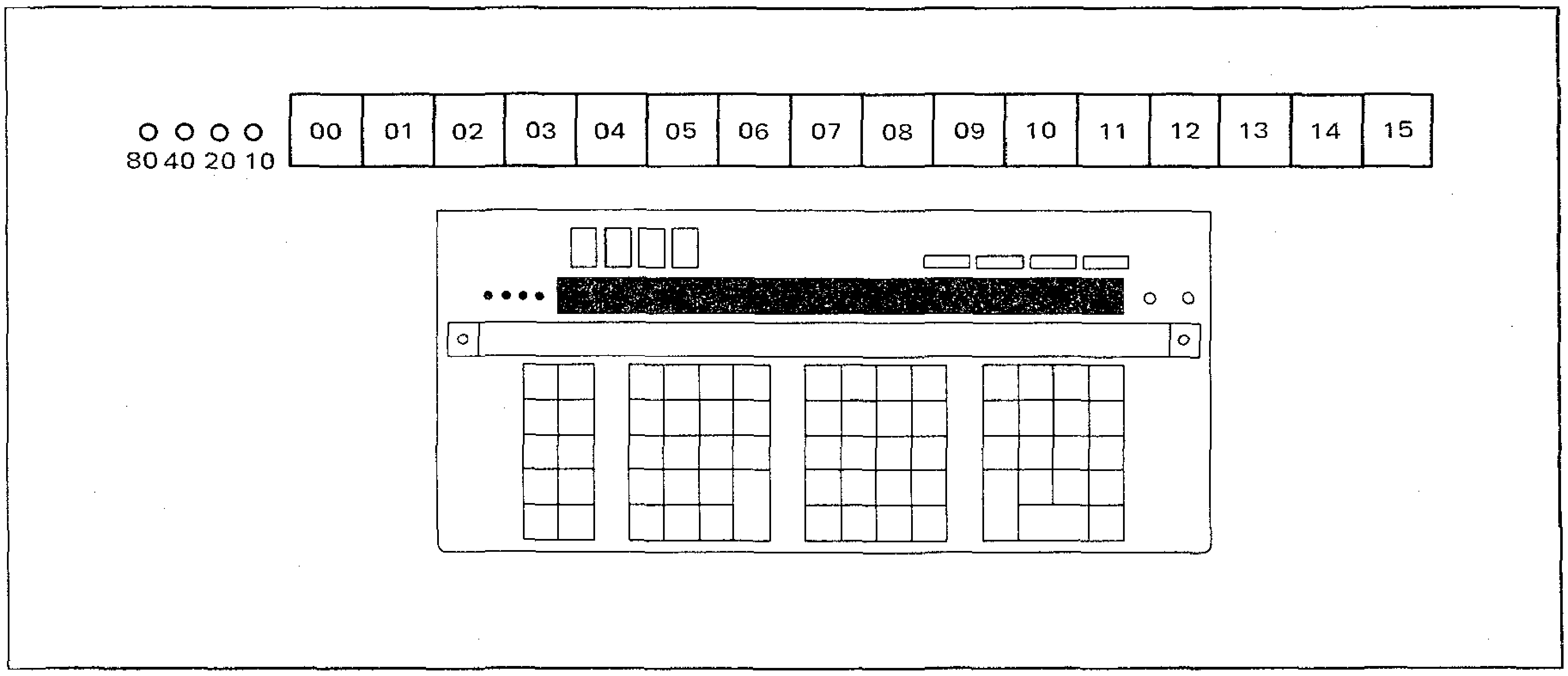
In addition to the X and Y work registers, the Wang 700 has up to 122 storage registers. Each register has a 12-digit mantissa with sign and a two-digit exponent with sign. The registers are numbered consecutively from 000 to 121 and can be addressed both directly and indirectly for maximum convenience. Numbers are stored from and recalled to the X-Register. Each register can be used to add, subtract, multiply and divide. Any number in storage can be exchanged or swapped with any number in the X-Register.

DIRECT ADDRESSING

Direct addressing of registers requires a two-step command. The first keystroke indicates the operation (i.e., to Store, Recall, Add, Subtract, Multiply, Divide, or Exchange). The second keystroke indicates the register in which the operation is to be performed. To store a number, simply index the control key **STORE DIRECT** followed by a second keystroke identifying the register number.

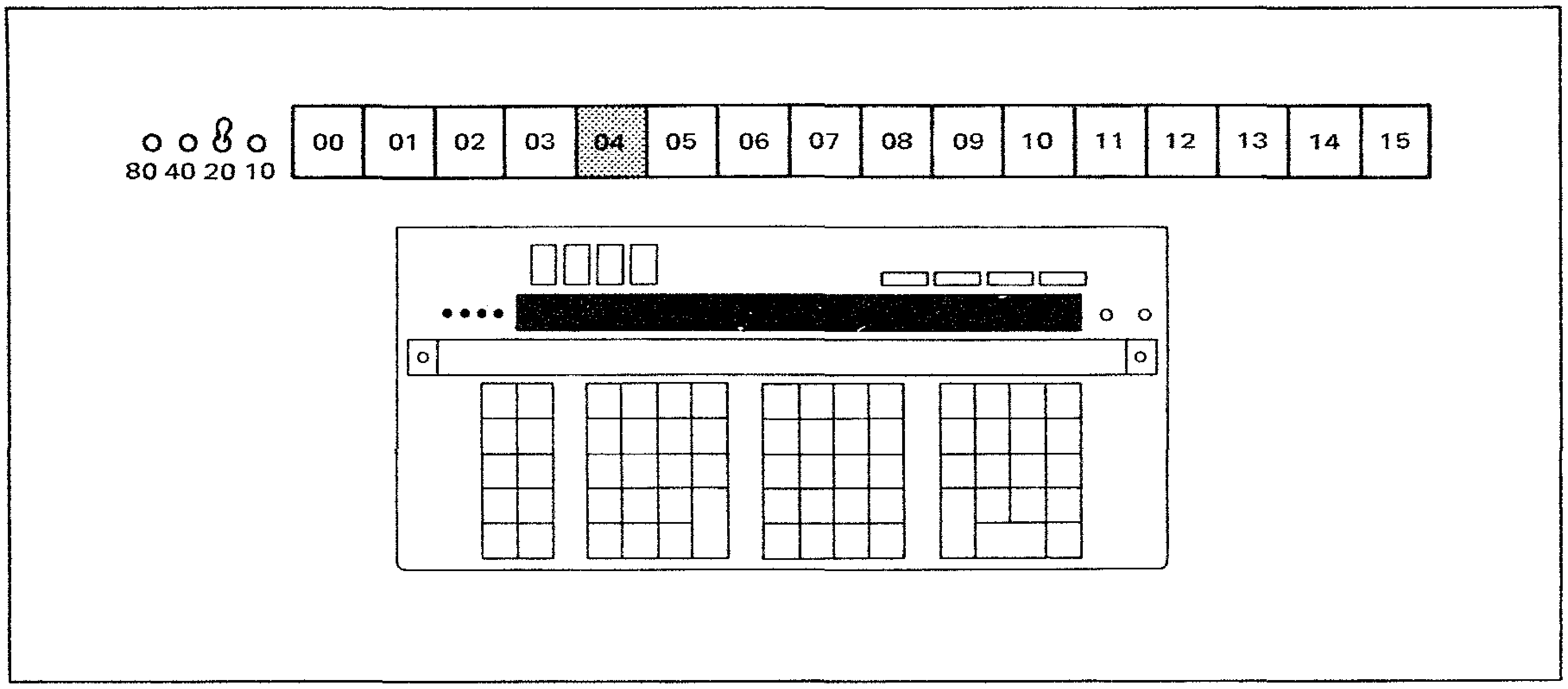
TOGGLE SWITCHES AND SPECIAL FUNCTION KEYS

Each register is represented by a combination of toggle switch settings and special function keys.



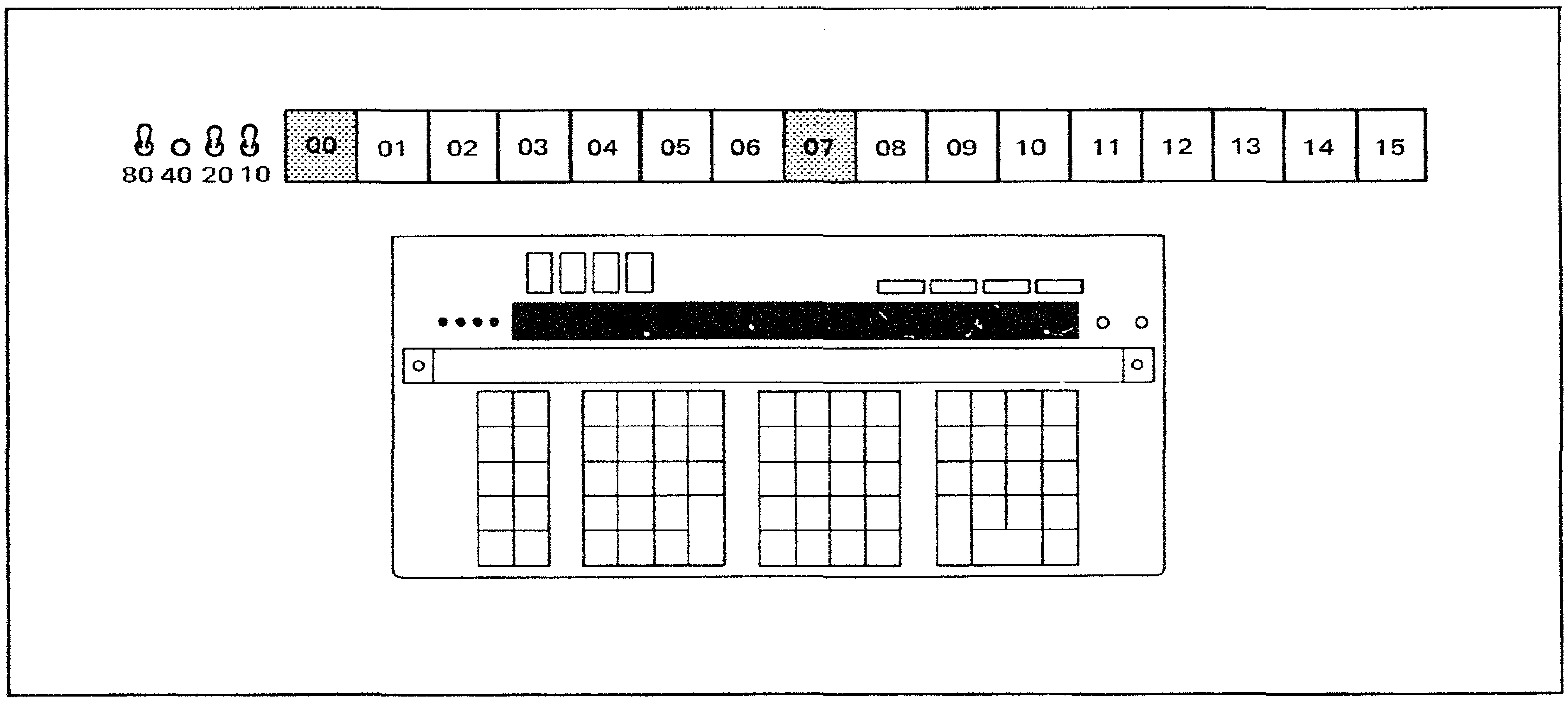
The toggle switches are set to the OFF (down) position. When the toggle switches are in the down position, the special function keys designate the registers 000 to 015. The 4 toggle switches are labeled 80, 40, 20 and 10. When one of these toggle switches is switched to the ON (up) position and a special function key is indexed, the register designated is the sum of the values of the toggle switches and the special function key.

(1)



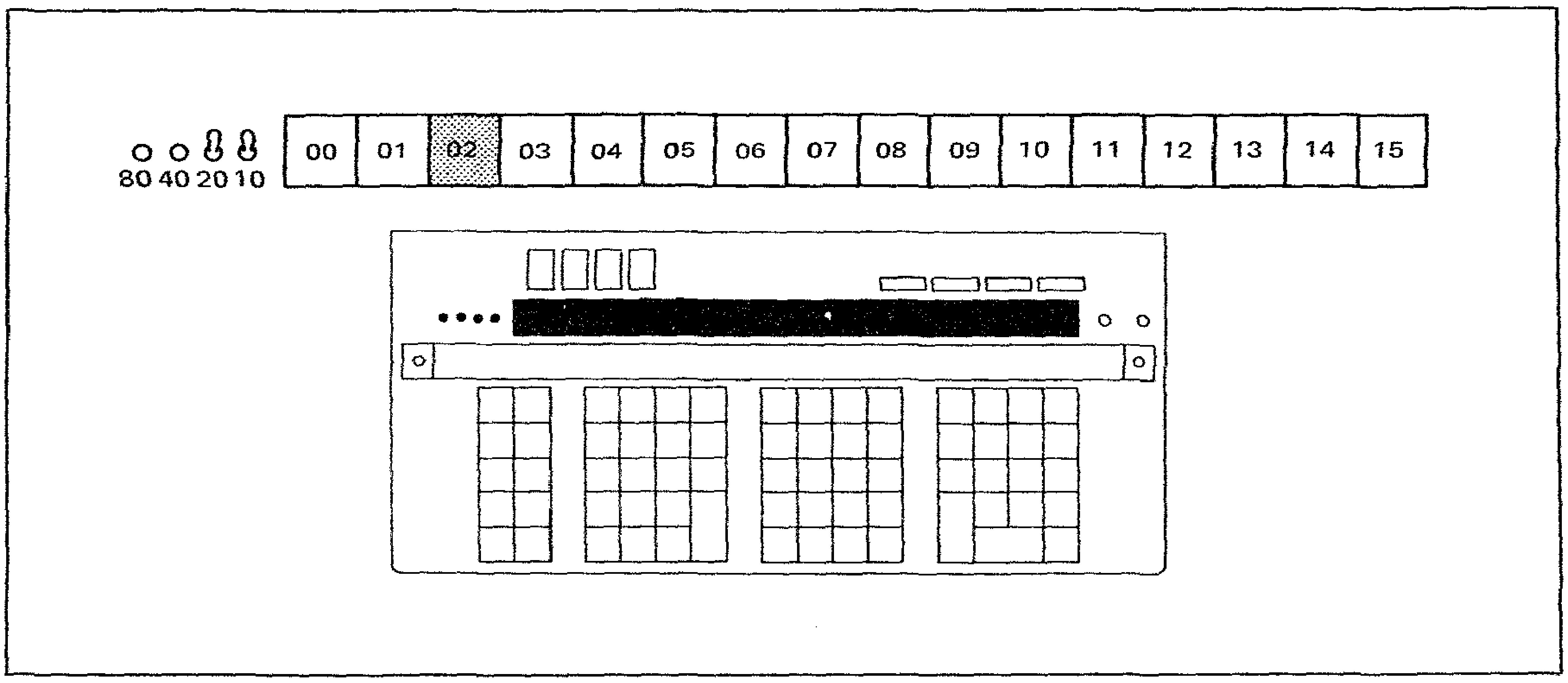
Depressing the special function key 04 while switch setting (20) is flicked ON designates register 24 (20 + 4).

(2)



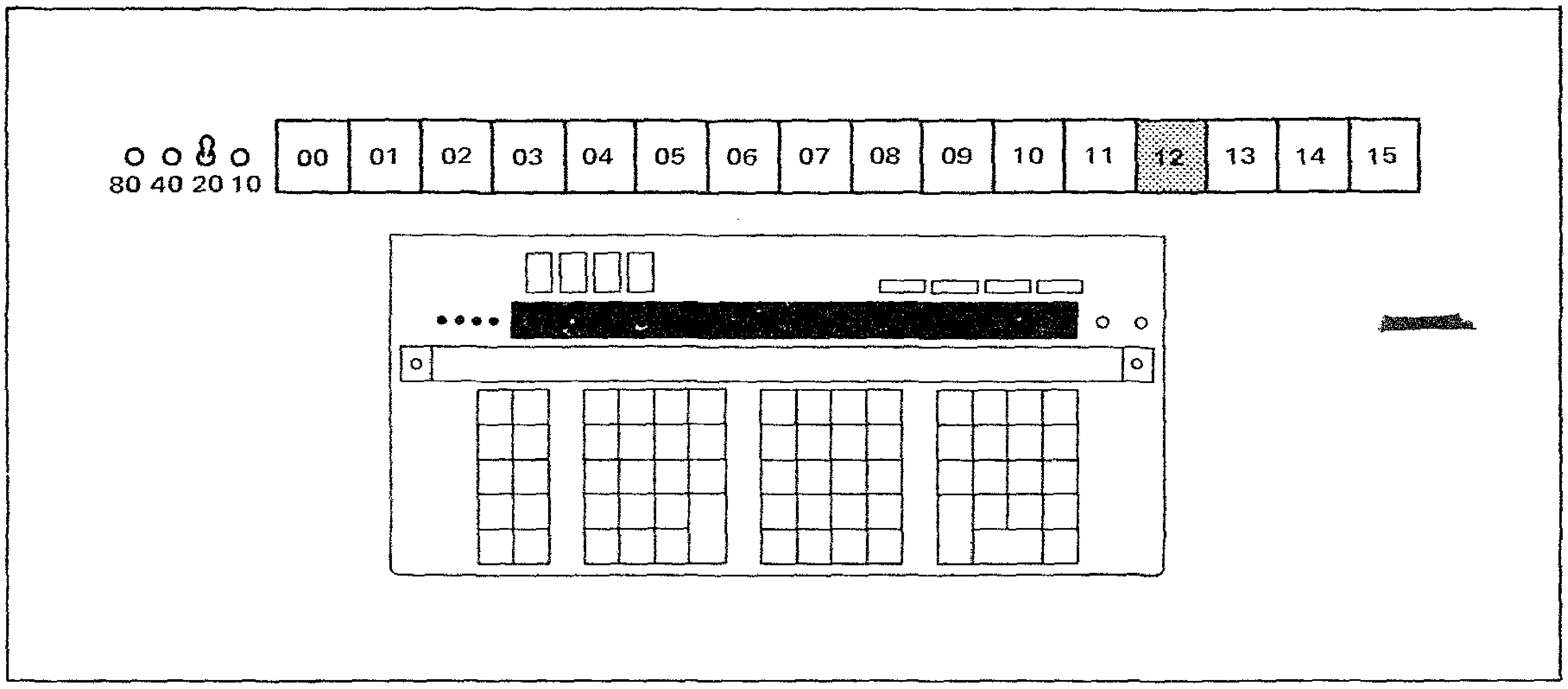
Notice the toggle switch setting. When the **07** key is indexed the register designated is 117 (80 + 20 + 10 + 7). When the **00** key is indexed the register designated is 110 (80 + 20 + 10 + 0).

(3) Designate Register 32 in two different ways:



One way of doing this would be to set toggle switches 20 and 10 to the ON (up) position and press the special function key **02**.

Another way is the following:



Set toggle switch 20 in the up position and press the **12** key. Notice both combinations 20 + 10 + 2 and 20 + 12 designate register 32. Thus, different combinations of toggle switch settings and special function keys can be used to identify a particular register. However, in

LEARN MODE the program code designating Register 32 would be 0302 or 0212, depending on which method was used.

STORE DIRECT

To store a number in a register, simply index the number into the X-Register, press the **STORE DIRECT** key followed by the register number.

EXAMPLE 1: Store π^2 into register 14

* Toggle switches down

Index π x^2 STORE DIRECT 14

π^2 is now stored in register 14 and is still displayed in the X-Register.

EXAMPLE 2: Store .57 x 10^{18} into Register 32

*Toggle switches 20 and 10 UP

Index 5 7 SETEXP 1 8

STORE DIRECT 02

.57 x 10^{18} is now stored in Register 32 and is still displayed in X.

***NOTE**

For problems requiring less than 17 storage registers and for general usage, the toggle switches are kept in the OFF (down) position and the Special Function Keys are used to address Registers 000 to 015.

RECALL DIRECT

RECALL DIRECT recalls the number from the designated register into the X-Register. The number appears in the X-Register and also remains in the storage register. The sequence of steps to follow is the same as with **STORE DIRECT**.

EXAMPLE: Recall π^2 from register 14

Index RECALL DIRECT 14

π^2 appears in the X-Register and is still in storage register 14.

⇄ DIRECT,

The ⇄ DIRECT key is a handy command which allows the operator to exchange a number in the X-Register with a number in any of the storage registers. The command simply swaps the values of the X-Register and the internal register. Again the sequence of steps to follow is ⇄ DIRECT followed by the desired register.

EXAMPLE: Suppose 27.8 is in the X-Register and π^2 is in Register 14. To store 27.8 in Register 14 and recall π^2 to the X-Register in one operation:
Index ⇄ DIRECT 14

What happens if the same operation is performed again?

ADD, SUBTRACT, MULTIPLY, AND DIVIDE DIRECT (The X and Y Registers Remain Unchanged.)

In addition to storing a 12-digit mantissa and a 2-digit exponent, the registers can be used as accumulators to add, subtract, multiply and divide. With each of these operations **the result is stored in the designated register** and **the X-Register and Y-Register remain unchanged.**

The four arithmetic operations are:

+DIRECT	Adds number in **X-REGISTER** to value stored in register designated by next keystroke. The X and Y Registers remain unchanged.
–DIRECT	Subtracts number in **X-REGISTER** from value stored in register designated by next keystroke. The X and Y Registers remain unchanged.
XDIRECT	Multiplies number in **X-REGISTER** by value stored in register designated by next keystroke. The X and Y Registers remain unchanged.
÷DIRECT	Divides number in **X-REGISTER** into number stored in register designated by next keystroke. The X and Y Registers remain unchanged.

A simple example will illustrate how each of these commands works.

EXAMPLE: Perform the following in Register 001

$$\frac{(13 \text{ x}2) + 4}{3} - 3 = 7$$

1.	1 3 STORE DIRECT 01	Places 13 in Register 01 and the X-Register
2.	2 X DIRECT 01	This sequence of steps places the product equal to 26 in Register 01 and 2 remains unchanged in the X-Register.
3.	4 +DIRECT 01	Adds 4 to the Answer. 30 is now in Register 01, 4 is in X-Register.

4.	3 ÷DIRECT 01	Divides result by 3 putting 10 in Register 01, 3 remains in X-Register.
5.	– DIRECT 01	Since 3 is in X when the command is given, 3 is subtracted from 10 putting 7 in Register 01, 3 in X-Register.
6.	RECALL DIRECT 01	Recalls final answer to X. = 7

The fact that the result is put in the storage register rather than the X-Register can be extremely useful if we are using a constant multiplier or divisor.

INDIRECT ADDRESSING

In addition to providing direct access to the internal storage registers, the Wang 700 offers an indirect mode of address. Both display registers are utilized for indirect addressing. The Y-Register designates the register being addressed. As with direct addressing, the X-Register is used as the work register. The command is performed on the number in X and the result is placed in the internal storage register.

Indirect addressing is a valuable programming tool for saving program steps, especially in repetitive matrix-type operations. Remember, indirect addressing requires only one step – the operation itself. The register on which the operation is performed is identified by the number in Y.

INDIRECT KEYS

The indirect commands are identical to those used in direct addressing. They consist of the following:

KEY	OPERATION
STORE INDIRECT	Stores number in X into Register designated by number in Y.
RECALL INDIRECT	Recalls number to X from register designated in Y. Number also remains in register.
⇄ INDIRECT	Swaps number in X with number in register designated by Y.
+ INDIRECT	Adds number in X to number in register designated in Y. The sum is placed in internal register. Number in X remains unchanged.

– INDIRECT	Subtracts number in X from number in register designated in Y. The difference is placed in internal register. Number in X remains unchanged.
X INDIRECT	Multiplies number in X by number in register designated in Y. The product is placed in internal register. Number in X remains unchanged.
÷ INDIRECT	Divides number in X into number in register designated in Y. The quotient is placed in internal register. Number in X remains unchanged.

The following example illustrates how each of these commands would be used.

Example

Perform the following in Register 002 using Indirect mode of address.

$$\left[\frac{7\,(5.8) - 7.2}{3}\right]^{2} + 3 = 126.951111110$$

KEY	OPERATION
2 ↑	Places the register number in Y (The register number is usually computed in the program.)
7 ST INDIR	Stores 7 in register 002. The value is now in both register 002 and the X Register.
5 . 8 X INDIR	Multiplies the value (7) in Register 002 by 5.8,putting the result in 002 and 5.8 remaining in X.
7 . 2 – INDIR	Subtract 7.2 from the value in Register 002 and places result in Register 002. 7.2 remains in X.
3 ÷ INDIR	Divides the value in Register 002 by 3. The result is put in Register 002 and 3 remains in X.
⇄ INDIR	Exchanges $\frac{7(5.8) - 7.2}{3}$ in Register 002 with 3 in the X-Register.
x^2	Squares the value in X.

<u>+ INDIR</u>	Adds $\left[\frac{7(5.8) - 7.2}{3}\right]^2$ to 3 in Register 002. The result is placed in Register 002 and $\left[\frac{7(5.8) - 7.2}{3}\right]^2$ remains in X.
<u>RECALL INDIR</u>	Recalls final answer = 126.951111110

ADVANTAGES OF INDIRECT ADDRESSING

Two advantages of the indirect mode of addressing are:

1. It requires only one keystroke to perform the indicated operation.
2. By constructing a loop, a given program step sequence can operate on many different sets of registers. A saving of many program steps can result from this technique.

Figure 1 is a simple program which illustrates the advantage of indirect addressing. The program stored 0 in the first 100 registers. Using direct access a minimum of 200 steps would be required. (Two keystrokes per register – **STORE DIRECT** followed by each register number.) In contrast, this program requires only 13 steps to accomplish the same thing. A savings of 187 steps!

FIGURE 1

OPERATING PROCEDURE: <u>PRIME</u> <u>GO</u>

STEP	KEY	CODE
000	MARK	0408
1	0700	0700
2	0	0700
3	ST IND	0504
4	1	0701
5	+	0600
6	1	0701
7	0	0700
8	0	0700
9	SKIP Y=X	0509
10	SEARCH	0407
11	0700	0700
12	STOP	0515

RECALL RESIDUE

The **RECALL RESIDUE** key is a unique Wang 700 feature which can be of great value to users who need greater than 12 digit accuracy. The **RECALL RESIDUE** key gives the user the option of double precision arithmetic for addition, subtraction, multiplication, and division performed in any of the storage registers or the X and Y registers. By indexing the **RECALL RESIDUE** key directly after performing one of these operations, another 12 digits of accuracy is acquired.

ADDITION, SUBTRACTION, MULTIPLICATION

When the **RECALL RESIDUE** key is indexed after performing an addition, subtraction or multiplication, a residue is displayed in the X-Register, which if added to the first 12 digits of the result, gives an additional 12 digits of accuracy. Examples are given to show how the **RECALL RESIDUE** key is used for addition, subtraction, and multiplication.

EXAMPLE 1:

ADD	OPERATION ON 700	DISPLAY
5024873058.28	5 0 2 4 8 7 3 0 5 8 . 2 8 ↑	+5024873058.28 +6.8520987
+ 6.8520987 5024873065.1320987	6 . 8 5 2 0 9 8 7	
	±	+.502487306513 +10 +6.8520927
	RESIDUE	+.502487306513 +10 +.209870000000 −02

By indexing the **RECALL RESIDUE** key, the significant digits which would ordinarily be lost in the shifting process are retained. The final result is always the algebraic sum of the values displayed.

In subtraction, however, the residue might be opposite in sign to the answer. This should not cause any difficulty since the residue is always algebraically added to the result.

EXAMPLE 2:

SUBTRACT	OPERATION ON 700	DISPLAY
5024873058.28	5 0 2 4 8 7 3 0 5 8 . 2 8 ↑	+5024873058.28 +6.8520987
− 6.8520987 5024873051.4279013	6 . 8 5 2 0 9 8 7 −	+.502487305143 +10 +6.8520987

	RESIDUE	+.502487305143 +10 −.209870000000 −02

In this example, the residue is opposite in sign to the result. If these two numbers are added together, the correct result is generated. An easy way of performing this addition is to decrease the 12th digit of the result by 1 (.502487305143 becomes .50248730512), subtract each digit of the residue from 9 so .20987 becomes .79012, and add 1 to the last significant digit (.79013).

Multiplication works the same way as addition.

EXAMPLE 3:

MULTIPLY	OPERATION ON 700	DISPLAY
31415.9254998 .728645297326	3 1 4 1 5 . 9 2 5 4 9 9 8	+31415.9254998 +.728645297326
	↑	
	. 7 2 8 6 4 5 2 9 7 3 2 6	+22891.0663764 +.728645297326
The answer is	X	
22891.0663765732361535348	RESIDUE	+22891.0663765 +.732361535348 −07

The first twelve digits of the product are in Y; the last 12 digits are in X.

DIVISION

Using the **RECALL RESIDUE** key in division is slightly different from addition, subtraction, and multiplication. In division, indexing the **RECALL RESIDUE** key gives us a remainder. Using this remainder and the original divisor, 12 more digits of accuracy can be obtained by performing the division again. Study the following example illustrating the technique:

EXAMPLE 4:

DIVIDE $\frac{22}{7}$	OPERATION ON 700	DISPLAY
3.14285714285 7 /220000000 21 10 7 30 28 20 14 60 56 40 35 50 49 10 7 30 28 20 14 60 56 40 35 Remainder ← $\boxed{5}$	22 ↑ 7 ÷ RESIDUE	+22.000000000 +7 +3.14285714285 +7.00000000000 +3.14285714285 +.500000000000 –11

The +.500000000000 – 11 displayed in X after the **RECALL RESIDUE** key is pressed indicates a remainder of 5 after the first 12 digits of the quotient are generated. Notice the proper decimal position is retained (i.e., .5 x 10^{-11}). Since the decimal position is retained automatically, the original divisor should be expressed with the decimal point in the left most position and an exponent value of 0 before performing the second division. Thus, .7 is divided into the remainder .5 x10^{-11} and 12 more digits of the quotient are generated. To preserve the first 12 digits of the quotient, the second division is performed in Register 000.

Since the remainder is now in X

STDIR 00
7 SETEXP* ÷DIR 00
RE DIR 00

*This command automatically aligns the decimal point and exponential value of the divisor.

Read .714285714285 – 11 in the X-Register which if added to 3.14285714285 yields 24 digit accuracy for 22/7. If greater accuracy is desired, simply touch the **RECALL RESIDUE** key to obtain the remainder (.5000000000000 – 23) and repeat the process.

This example illustrates the fact that the **RECALL RESIDUE** key performs the same function when any of the 120 internal registers are used to add, subtract, multiply and divide. The **RECALL RESIDUE** key is NOT limited to use solely with the X and Y registers. **IT SHOULD ALSO BE NOTED THAT THE RESIDUE MUST BE SAVED AFTER EACH OPERATION IF IT IS TO BE USED IN FURTHER CALCULATIONS.**

WRITE COMMANDS

The 701 Output Writer provides the user with completely formatted alpha-numeric output of his calculated results.

NUMERIC output consists of a two-step command. The WRITE key followed by a format command will print the contents of the X-Register.

The format command specifies the number of digits to be printed out before and after the decimal point

EXAMPLE

WRITE
02 03

The HIGH ORDER digit of the code specifies the number of digits before the decimal point.

The LOW ORDER digit of the code specifies the number of digits after the decimal point.

The above command would print two digits before the decimal point and three digits after the decimal point.

An option to always print in modified scientific notation is available.

EXAMPLE

```
Display:                      +.12345678123 – 40
Command:                      WRITE
                              0015
Output will appear as:        .123456789123ex – 40
```

ALPHABETIC output can be printed under program control by using the **WRITE ALPHA** command. Indexing the **WRITE ALPHA** key places the 700 in alpha mode so that alpha characters can be printed. The **END ALPHA** command takes the 700 out of alpha mode.

EXAMPLE

```
    WRITE ALPHA               (Places 700 in alpha mode)

          H – 0101
          E – 0205
          L – 0109
          L – 0109
          O – 0209

    END ALPHA                 (Takes the 700 out of alpha mode)
```

The above example would print the word "**HELLO**."

Other control commands such as shifting to upper and lower case, carriage return, line feed, spacing, backspace, and tabulation are all available on the Output Writer. All these features are discussed in the 701 OUTPUT WRITER MANUAL.

GROUP 1–GROUP 2

These two keys are reserved for addressing optional peripheral equipment.

SECTION III
PROGRAMMING

CODING

All programmed operations are represented by a 4-digit code. A list of the keyboard operations and their respective codes is given below:

700 PROGRAM CODES

CODE	KEY	CODE	KEY
0400	+ DIRECT	0600	+
0401	– DIRECT	0601	–
0402	x DIRECT	0602	x
0403	÷ DIRECT	0603	÷
0404	STORE DIRECT	0604	↑
0405	RECALL DIRECT	0605	↓
0406	⟳DIRECT	0606	⟲
0407	SEARCH	0607	|X|
0408	MARK	0608	INTEGER X
0409	GROUP 1	0609	π
0410	GROUP 2	0610	$\text{Log}_{10}X$
0411	WRITE	0611	$\text{Log}_e X$
0412	WRITE ALPHA	0612	$\sqrt{X}$
0413	END ALPHA	0613	10^x
0414	STORE Y *	0614	e^x
0415	RECALL Y *	0615	1/x
0500	+ INDIR	0700	0
0501	– INDIR	0701	1
0502	x INDIR	0702	2
0503	÷ INDIR	0703	3
0504	STORE INDIR	0704	4
0505	RECALL INDIR	0705	5
0506	⟳INDIR	0706	6
0507	SKIP if $Y \geqslant X$	0707	7

*ENTERED BY TOGGLE SWITCH SETTING

0508	SKIP if Y < X	0708	8
0509	SKIP if Y = X	0709	9
0510	SKIP if ERROR	0710	SET EXP
0511	RETURN	0711	CHANGE SIGN
0512	END PROG	0712	DECIMAL POINT
0513	LOAD PROG	0713	X^2
0514	GO	0714	RECALL RESIDUE
0515	STOP	0715	CLEAR X

The four-digit code consists of 2 halves: a high-order 2-digit number and a low-order 2-digit number.

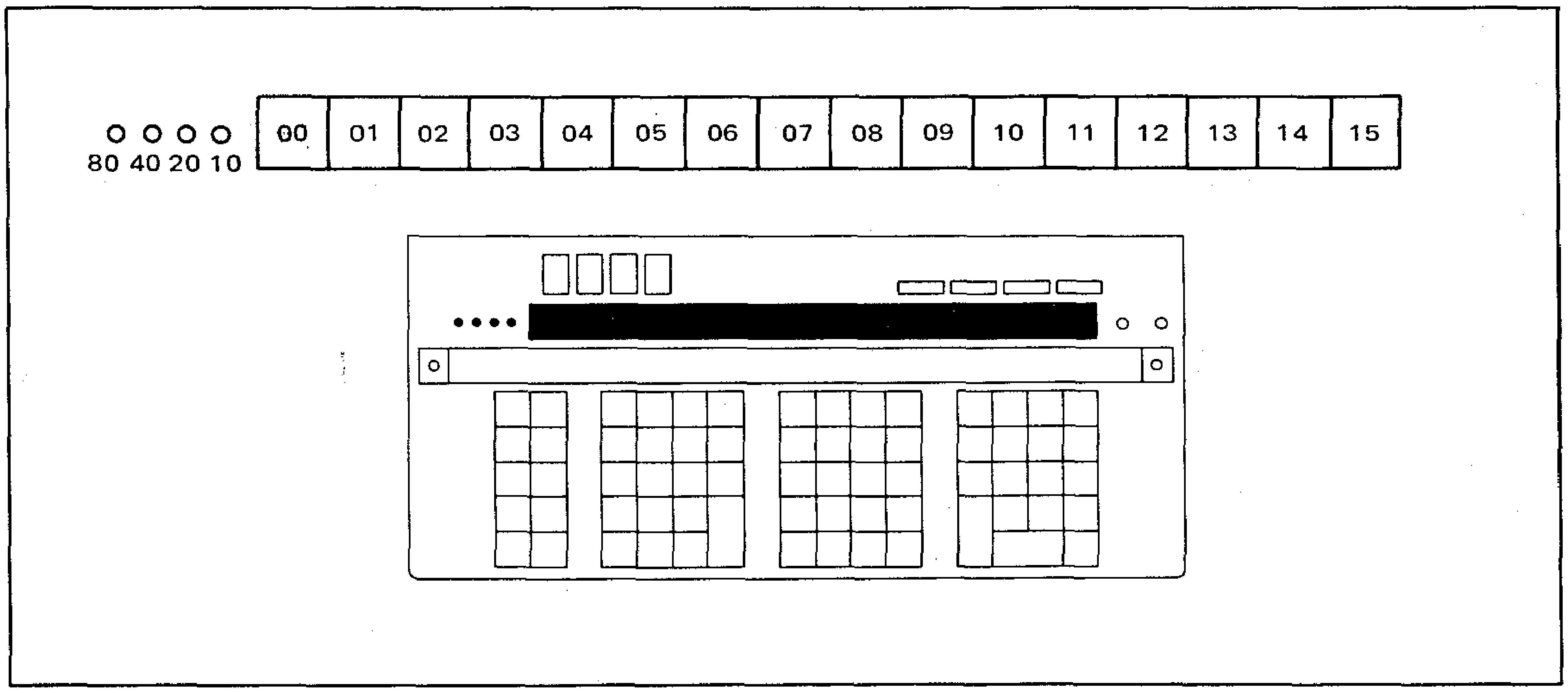
$$\underbrace{X\,X}_{\substack{\text{HIGH}\\\text{ORDER}}}\quad\underbrace{X\,X}_{\substack{\text{LOW}\\\text{ORDER}}}$$

Each of these halves can assume the values 00, 01, 02, . . . up to 15. Thus there are 16 different high and low-order digits and a total of 16 x 16 = 256 codes.

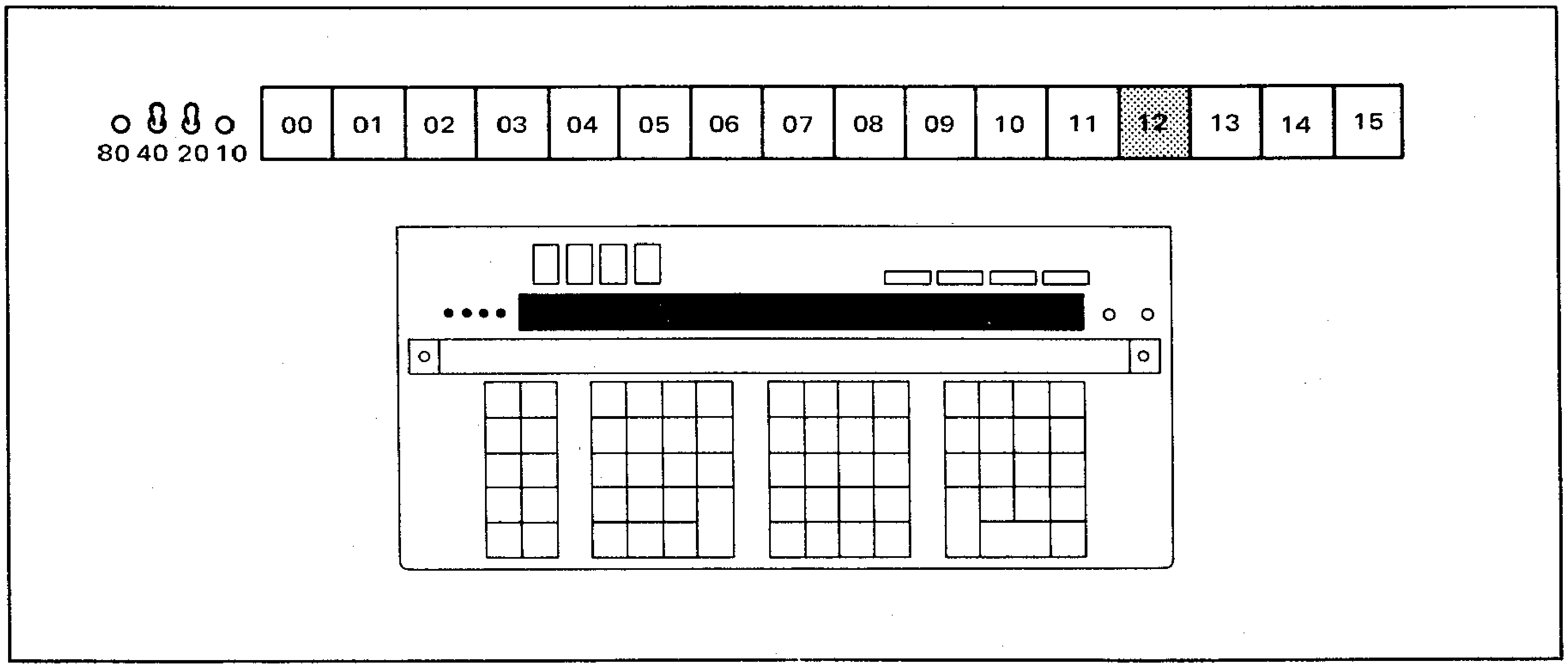
The 64 codes used in the above table are set aside for the keyboard operations. They consist of all possible combinations that can occur when the high-order digit assumes the values 04, 05, 06 and 07 and the low-order digit assumes the values 00 to 15 – a total of 64 codes (16 combinations are in each of the 4 categories).

GENERATING A CODE USING SPECIAL FUNCTION KEYS AND TOGGLE SWITCHES

While this procedure is not recommended for any of the "operation keys," any legal code can be generated using the toggle switches and the special function keys. The special function keys are used to define the low-order digit and a combination of toggle switches is used to define the high order digit.



The toggle switches are labeled 80, 40, 20, and 10 for convenience in selecting the data storage registers discussed in Section II. **THEY CAN ALSO BE VISUALIZED AS REPRESENTING THE NUMBERS 08, 04, 02,AND 01 FOR THE PURPOSE OF GENERATING THE HIGH-ORDER DIGIT OF ANY LEGAL CODE.** When a special function key is indexed, the operation executed by the calculator is the command whose high-order digit is defined by the setting of the toggle switches and whose low-order digit is the special function key indexed.



If the toggle switches are set as in the above figure and the special operation key **12** is indexed, the square root of the number in the X-Register will be generated since the code for square root is 0612. Naturally, the square root of a number would rarely be found using this technique, however, this example is included to explain how to generate any of the 256 codes. This technique is used most often with the Store Y and Recall Y commands.

CORE MEMORY

Core Memory is organized into 121 or 122 data registers numbered consecutively from 000 to 121 or 122. Registers 000 – 119 are used for storing either program steps or data. 16 program steps occupy 2 data-storage registers. Register 120 and 121 are used exclusively for data storage .(700B data register 121 not available.)

As stated previously, each programmed operation is represented by a four-digit code. The four-digit code consists of two halves: a high-order two-digit number and a low-order two-digit number.

$$\underbrace{XX}_{\substack{\text{HIGH}\\\text{ORDER}}}\quad\underbrace{XX}_{\substack{\text{LOW}\\\text{ORDER}}}$$

The program code for $\sqrt{x}$ is $\underbrace{06}_{\substack{\text{HIGH}\\\text{ORDER}}}\quad\underbrace{12}_{\substack{\text{LOW}\\\text{ORDER}}}$

CORE STORAGE

REG NO.			
000 001	959		944
002 003	943		928
004 005	927		912
006 007	911		896
008 009	895		880
010 011	879		864
012 013	863		848
014 015	847		836
016 017	831		816
018 019	815		800
020 021	799	HIGH ORDER LOW ORDER	784
022 023	783		768
024 025	767		752
026 027	751		736
028 029	735		720
030 031	719		704
032 033	703		688
034 035	687		672
036 037	671		656
038 039	655		640

PROGRAM STEP NO.

REG NO.			
040 041	639		624
042 043	623		608
044 045	607		592
046 047	591		576
048 049	575	HIGH ORDER LOW ORDER	560
050 051	559		544
052 053	543		528
054 055	527		512
056 057	511		496
058 059	495		480
060 061	479		464
062 063	463		448
064 065	447		432
066 067	431		416
068 069	415		400
070 071	399		384
072 073	383		368
074 075	367		352
076 077	351		336
078 079	335		320

PROGRAM STEP NO.

REG NO.			
080 081	319		304
082 083	303		288
084 085	287		272
086 087	271		256
088 089	255		240
090 091	239		224
092 093	223		208
094 095	207		192
096 097	191		176
098 099	175		160
100 101	159		144
102 103	143		128
104 105	127	HIGH ORDER LOW ORDER	112
106 107	111		096
108 109	095		080
110 111	079		064
112 113	063		048
114 115	047		032
116 117	031		016
118 119	015 014 013 012 011 010 009 008 007 006 005 004 003 002 001		000

PROGRAM STEP NO.

A Program code step occupies two digits of storage, one digit in each of two adjacent registers; the high-order digit of a code occupying one register; the low-order digit the other register.

Program steps 000 to 015 occupy registers 118 and 119. The following routine to add 2+2 is loaded into Registers 118+119 as illustrated.

STEP	KEY	CODE
000	2	0702
001	↑	0604
002	+	0600
003	STOP	0515
004	END PROG	0512

FIGURE 1

Registers

118												05	05	06	06	07	high order
119												12	15	00	04	02	low order
	015	014	013	012	011	010	009	008	007	006	005	004	003	002	001	000	

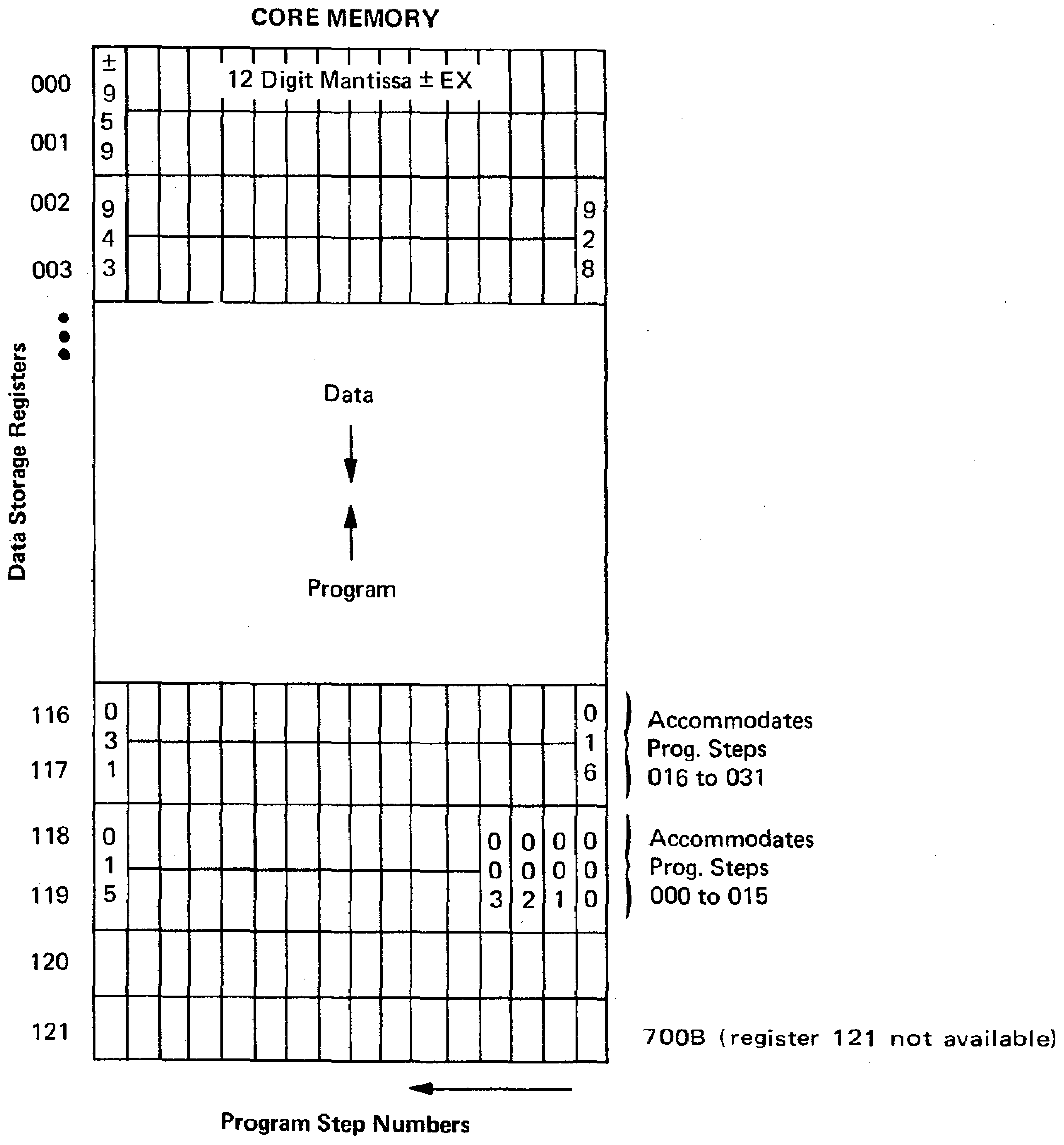
PROGRAM STEP NUMBERS

The high-order digit of the program code is loaded into Register 118, the low-order digit of the code in Register 119. Each pair of registers can accommodate 16 program steps. The program steps are numbered 000 to 959. Step 000 is located in Data Registers 119 and 118, Step 959 is located in Data Registers 000 and 001. (See Page 3-4) It shows exactly what program steps are located in each register. It is advisable to use registers 000, 001, 002 003, etc. for data storage and registers 119, 118, 117, etc., for program storage. In this way data will be stored in one end of core and program operations will be stored in the opposite end of core.

NUMBER OF REGISTERS OCCUPIED BY A PROGRAM

If a program is 7 steps long, 2 data registers are being utilized for storing the program. If the program is 35 steps long, 6 data registers are being used for program steps. To determine how many registers are being utilized:

1. Divide the number of program steps by 16.
2. Round the answer to the next whole number.
   Example:
   $\frac{33}{16}$ = 2.0625 becomes 3
3. Multiply the whole number by 2 to find the equivalent number of registers being used.



Example

Program of 88 steps occupies 12 registers

$$\frac{88}{16} = 5.5 \rightarrow 6\,(2) = 12 \text{ registers}$$

See Section V, Page 5-4 (for short program to perform this calculation.)

SECTION IV PROGRAMMING CONCEPTS

PROGRAMMING CONCEPTS

To exploit the full programming capability of the 700, a few basic programming concepts must be discussed. These are the concept of a branch, a subroutine, a loop and a decision.

Usually, the main part of a program advances one step at a time in a linear and continuous fashion. Each operation is performed consecutively one after the other. A program to evaluate the polynomial $y = 20x^2 + 5x + 7.2$ for different values of x would be:

OPERATION	CODE	REMARK
ST DIR	0404	Key in x
REG 00	0000	
x^2	0713	
↑	0604	
2	0702	
0	0700	
x	0602	$20x^2$ in y
5	0705	
X DIR	0402	
REG 00	0000	
RE DIR	0405	
REG 00	0000	
+	0600	$20x^2 + 5x$ in y
7	0707	
.	0712	
2	0702	
+	0600	$20x^2 + 5x + 7.2$ in y

Notice "the program" is simply the steps the user would perform if he were doing the calculation manually on the keyboard. However, the program needs some sort of command to tell the calculator where to start and where to end its calculation. This is the purpose of the SEARCH and MARK commands.

MARK AND SEARCH COMMANDS

Flags (names or marks) in a program are set by the **MARK** key followed by a second keystroke. To set a flag requires 2 keystrokes: **MARK** followed by any of the 256 legal codes. Thus, there are 256 different "names" or flags which can be used in a Wang 700 program. For the simple program we have written to evaluate $Y = 20x^2 + 5x + 7.2$, the number key 1 is used as a distinguishing flag. Thus, the program is preceded by the 2 keystrokes **MARK** 1. To end the calculation simply add a STOP command. The complete program thus becomes:

OPERATION	CODE	REMARK
MARK	0408	
1	0701	
ST DIR	0404	
REG 000	0000	
x^2	0713	
↑	0604	
2	0702	
0	0700	
x	0602	
5	0705	
X DIR	0402	
REG 00	0000	
RE DIR	0405	
REG 00	0000	
+	0600	
7	0707	
.	0712	
2	0702	
+	0600	
STOP	0515	

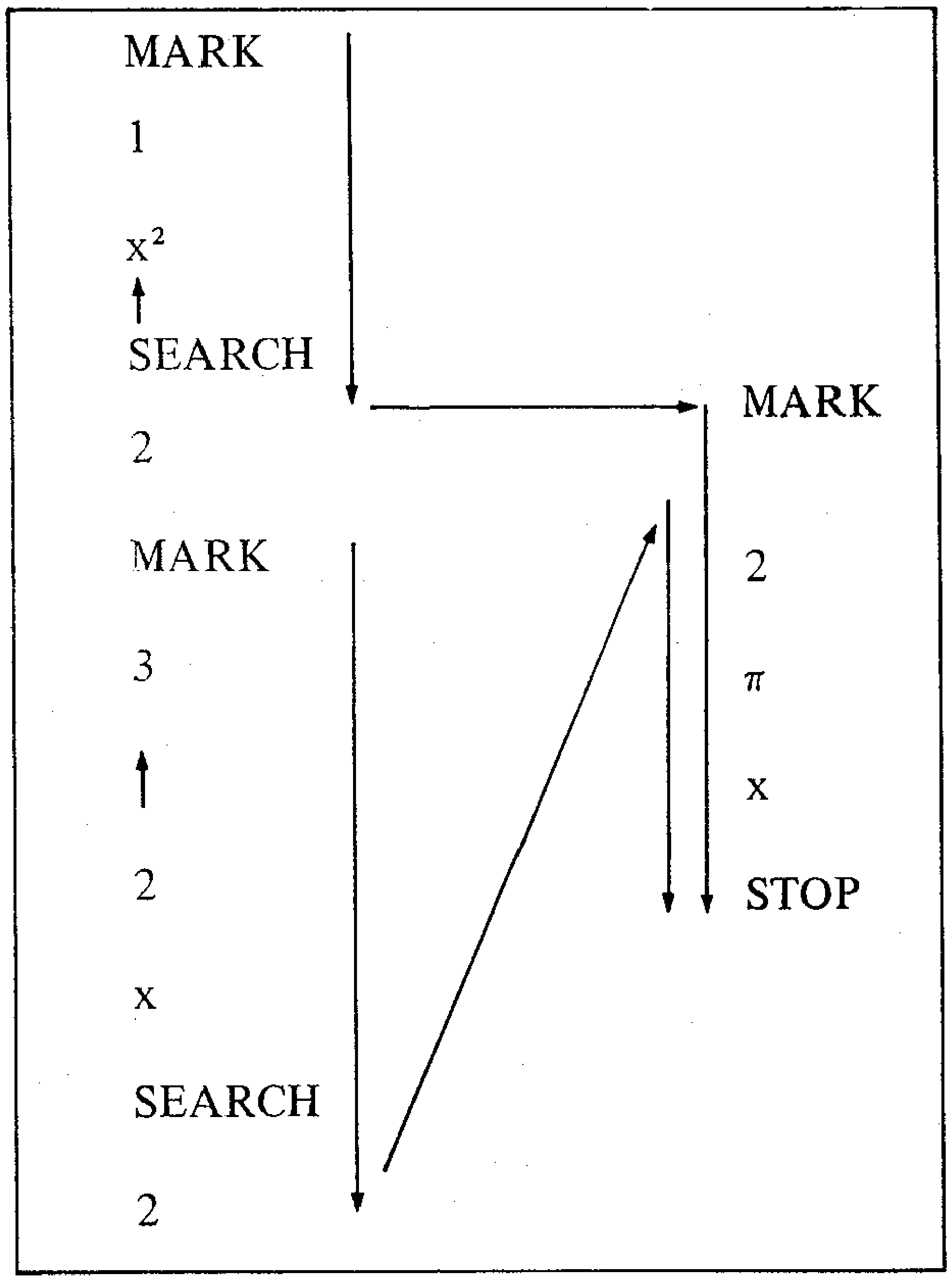
Flags tell the 700 where to start its calculations. They indicate the destination of a SEARCH command. In the SEARCH command, 2 keystrokes are required: SEARCH followed by a second keystroke which identifies the flag or mark to find. Thus, the operating procedure for the above program would be:

Key X; **SEARCH 1**

and the operations between **MARK 1** and the **STOP** command would be executed in sequence.

Generally, the numeric keys 0, 1, 2, . . . 9 are used as flags or markers for starting general programs. However, any programmable key on the 700 keyboard can be used as a name or marker. A program can start with a **MARK** e^X and to locate this mark, simply **SEARCH** e^X. It should be clear that when the **MARK** and **SEARCH** keys are indexed the calculator interprets the next keystroke as a name or flag and **not** as any other type of operation. When a **SEARCH X** command is given the 700 searches through core to locate the designated marker. If on scanning core it doesn't find the mark, the program stops and the **PROGRAM ERROR INDICATOR** goes on indicating there is no such mark in core.

The program for evaluating the polynomial follows a linear sequence of steps. The program executes step 000, then 001, then 002, and so on through to the last step. However, the Wang 700 does not have to follow a linear sequence of steps. It is possible for the 700 to start executing commands from step 025 and go through to step 052, then jump to step 075 ignoring all the commands between step 052 and 075. To break out of a linear sequence of steps and to jump about in a program is called branching. Both conditional and unconditional branching are possible on the 700. The **SEARCH** and **MARK** commands are used respectively for branching and for defining the destination of a branch.



The program on the preceding page evaluates A = πr^2 or C = $2\pi r$, depending on which steps are executed in the program.

A. To find A = πr^2 : Index r **SEARCH 1**
The program starts by squaring r and putting the result in Y, it then branches to **MARK** 2 ignoring all commands until it encounters the designated flag, and then multiplies r^2 by π for the final result in Y.

B. To find C = $2\pi r$: Index r **SEARCH 3**
The program ignores the commands preceding **MARK** 3 and starts by putting r in Y and multiplying it by 2. It then branches to **MARK** 2 ignoring all commands until it encounters the designated flag, and then multiplies 2r by π for the final result in Y.

As this program demonstrates, the SEARCH command can be part of a program, or can be keyed in by the operator, or both. In either case, upon encountering this command the program branches immediately to the designated mark. **MARK** and **SEARCH** commands can be located at any point or step in the program.

```
                 MARK

Initialization   1
                 0
                 ↑
                 ST DIR
                 REG 00
                 ST DIR
                 REG 01
Loop             MARK
                 0808
                 STOP
                 + DIR
                 REG 00      Σx
                 x²
                 + DIR
                 REG 01      Σx²
                 1
                 +           n
                 SEARCH
                 0808
Results          MARK
                 2
                 RE DIR
                 REG 00
                 STOP        Σx
                 RE DIR
                 REG 01
                 STOP        Σx²
```

OPERATING INSTRUCTIONS

1. PRIME; SEARCH 1
2. Key X, GO
   Repeat 2 for all x
3. SEARCH 2
   Read Σx in X
   Read N in Y
4. GO
   Read Σx^2 in X

The program on the preceding page further illustrates the idea of branching and introduces the important concept of looping. The program computes the statistical sums; Σx and Σx^2 for any number of x values. The first set of instructions initializes the registers by storing 0 in Y, Register 000, and Register 001. The second part of the program forms a loop which accumulates the Σx in Register 000, Σx^2 in Register 001, and the number of points entered in the Y-Register. The same operations are performed on each x-value. The program exits from the loop when a **SEARCH** 2 command is given. The final set of instructions recalls the answers to the display.

SUBROUTINE

Another idea closely related to branching is the concept of a subroutine. A subroutine is a part of a program (a sub-program) which appears several times within the overall program. Subroutine capability allows the program to branch to a specified routine, perform the calculations, and then return from where the program originally branched.

On the 700, a single keystroke is needed to branch to a subroutine. A set of 64 operation codes is reserved for this purpose. They consist of the 64 combinations which occur when the high-order digit of the 4 digit code assumes the values 00, 01, 02, and 03. A complete list of these codes is given in Table 1.

0000	0100	0200	0300
0001	0101	0201	0301
0002	0102	0202	0302
0003	0103	0203	0303
0004	0104	0204	0304
0005	0105	0205	0305
0006	0106	0206	0306
0007	0107	0207	0307
0008	0108	0208	0308
0009	0109	0209	0309
0010	0110	0210	0310
0011	0111	0211	0311
0012	0112	0212	0312
0013	0113	0213	0313
0014	0114	0214	0314
0015	0115	0215	0315

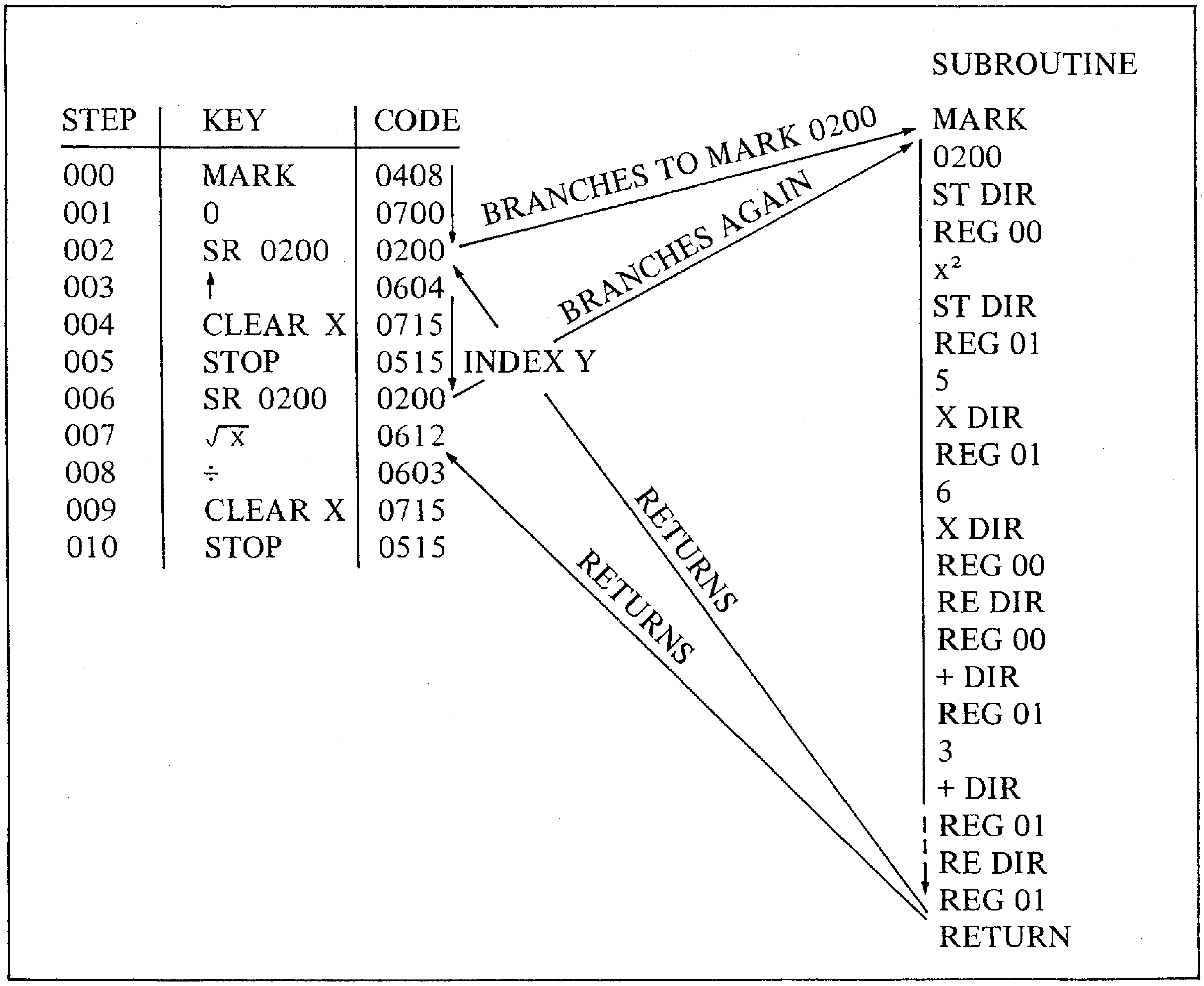
TABLE 1

EXAMPLE: Calculate the following for Z

$$Z = \frac{5x^2 + 6x + 3}{\sqrt{5y^2 + 6y + 3}}$$

OPERATING INSTRUCTIONS:

INDEX X SEARCH 0
Y GO
Read Z in Y



STEP	KEY	CODE
000	MARK	0408
001	0	0700
002	SR 0200	0200
003	↑	0604
004	CLEAR X	0715
005	STOP	0515
006	SR 0200	0200
007	$\sqrt{x}$	0612
008	÷	0603
009	CLEAR X	0715
010	STOP	0515

SUBROUTINE
MARK
0200
ST DIR
REG 00
x^2
ST DIR
REG 01
5
X DIR
REG 01
6
X DIR
REG 00
RE DIR
REG 00
+ DIR
REG 01
3
+ DIR
REG 01
RE DIR
REG 01
RETURN

When the first 0200 command is encountered at step 002, the program branches to **MARK 0200.** At the **RETURN** command, the program branches back to step 003 and continues on with the program. At the second 0200 command, the program again branches to the subroutine defined by **MARK 0200.** However, at the **RETURN** command the program branches back to step 007. There is no limit to the number of times a subroutine can be addressed and executed. The SR preceding the command in the KEY column is simply a mnemonic device indicating to the reader that a subroutine is being addressed.

It should be noted that the subroutine addressed through one of the 64 designated codes in Table 1 is preceded by a **MARK XXXX** of that same code and terminated by a **RETURN** command; otherwise, the calculator will not know when to return to the spot from which it originally branched.

The 64 codes listed in Table 1 do not necessarily have to define a subroutine. They can be used as regular marks and would then be addressed by the 2-step command **SEARCH XXXX.** However, it is generally considered wiser to reserve these codes exclusively for defining subroutines.

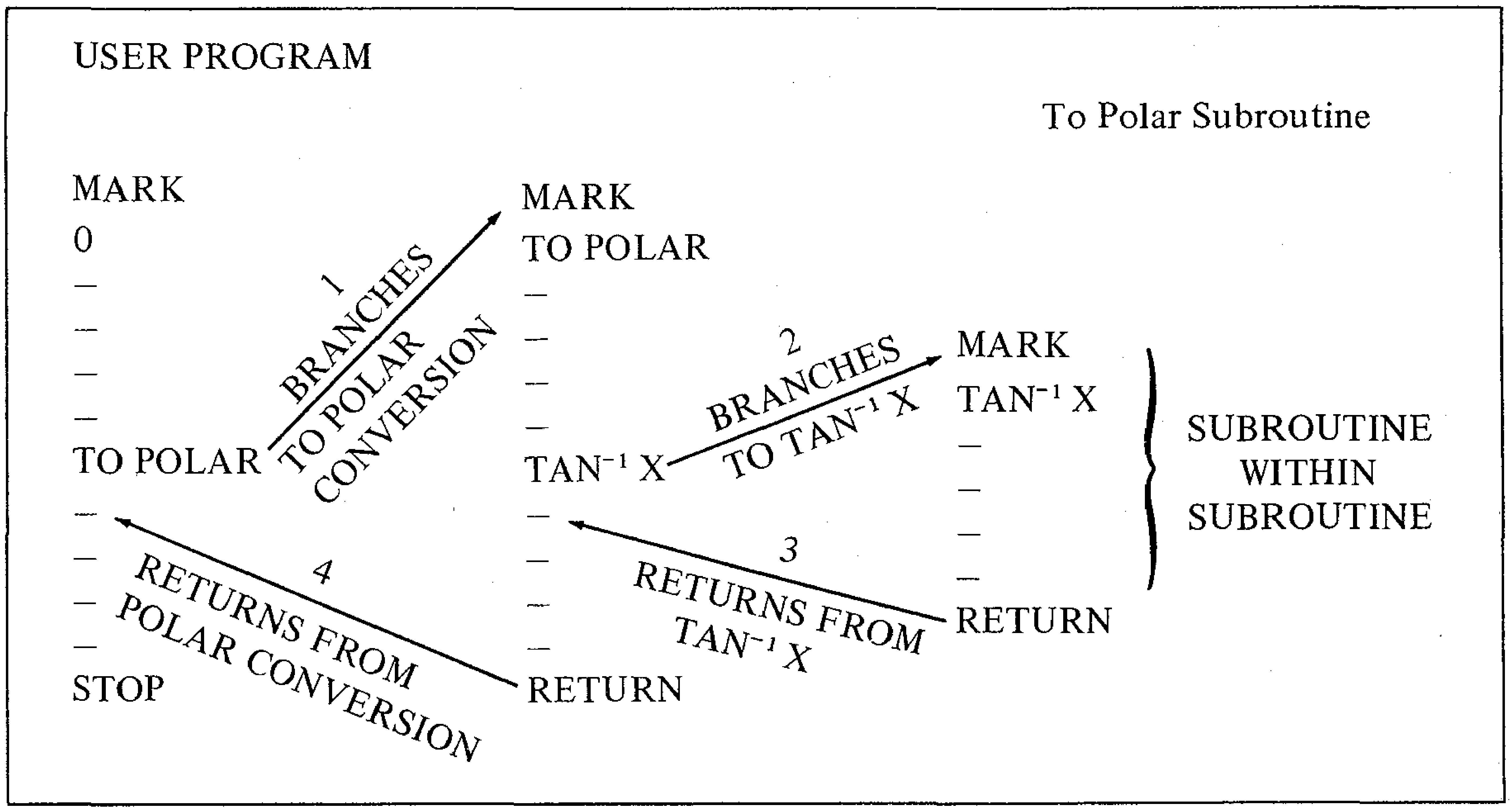
MULTI-LEVEL SUBROUTINES (Or a Subroutine within a Subroutine)

On the WANG 700, multi-level subroutines are possible. What does this mean? An example will best illustrate this concept. In the polar conversion in the TRIG PACK, the following formula is used to find

$$\theta = \tan^{-1} \frac{y}{x}$$

Therefore, the polar conversion subroutine addresses the TAN^{-1} X subroutine. This means the 700 must remember what step to branch back to after each of the two **RETURN** commands are executed.

The 700A is capable of remembering 2 return addresses. Thus, it has a double - level subroutine capability. 700B is capable of 5 return addresses, thus it has a five level subroutine capability. Figure 2 illustrates this concept graphically. The program branches to subroutine, it immediately branches to this routine, executes it and on encountering the **RETURN** command, branches back to the polar conversion routine which it continues to execute. When the second **RETURN** command is encountered, control branches back to the main program and the remaining steps are executed.



SECTION V
DECISION COMMANDS

DECISIONS

The Wang 700 has four decisions it can perform. They are used to check for the existence of certain conditions. If the condition is met, the program skips the next two steps. If the condition is not met, the program executes the next step. The four commands are **SKIP IF Y = X, SKIP IF Y ≥ X, SKIP IF Y < X,** and **SKIP IF ERROR.**

(1) Skip if Y = X

This command checks to see if the value in the Y-Register and X-Register are equal. If Y = X the program skips the next two steps. If Y does not equal X the program continues with the next step.

As a simple example:

Path for Y ≠ X	STEP	KEY	Path for Y = X
↓	000	MARK	↓
↓	001	0	↓
↓	002	SKIP Y = X	┐
Executes Steps 3 & 4 ↓	003	3	SKIPS STEPS 3 & 4
↓	004	STOP	│
	005	4	←
	006	STOP	↓

This program will put 3 in the X-Register if Y is not equal to X, and a 4 in X if Y = X.

> **NOTE**
>
> In testing for the condition Y = X, the programmer should keep in mind the necessity for absolute equality of the numbers in X and Y. A condition which is not ordinarily found in analytical cqmputations. Discrepancies often occur between the true value and the calculated value of a number.
>
> Illustration:
>
> Calculate $Y = \left[\frac{1}{3}\right] \times 3 \overset{?}{=} 1$

If a 1 is placed in X and the command **SKIP IF Y = X** is given, the calculator will treat the numbers as being unequal. Any good book on numerical analysis gives a full discussion on these discrepancies which occur in approximation theory.

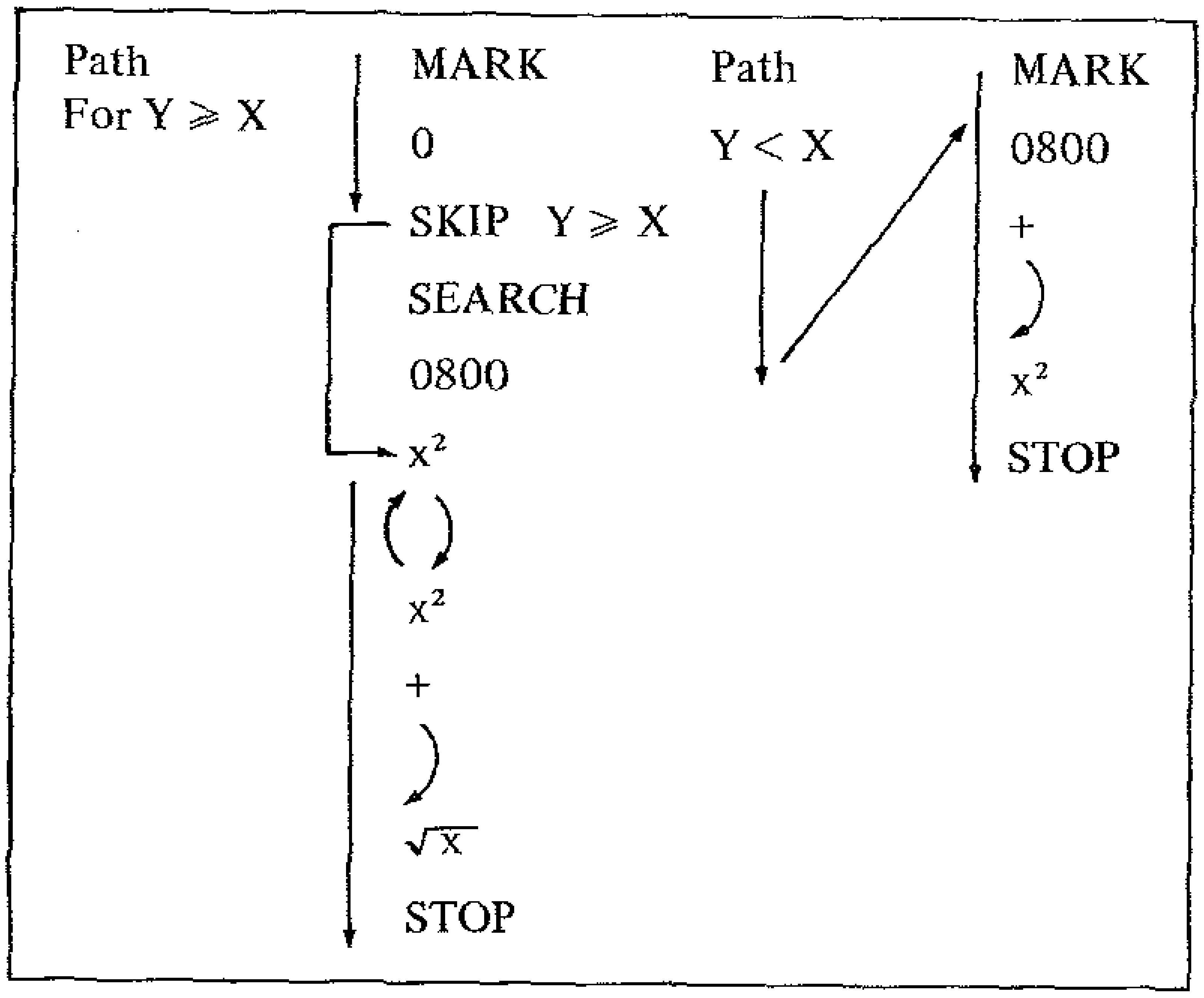
(2) Skip if Y ⩾ X

This command checks to see if the value in the Y-Register is equal to or greater than the value in the X-Register.

In the program below:

If Y ⩾ X the value $\sqrt{Y^2 + X^2}$ is calculated.

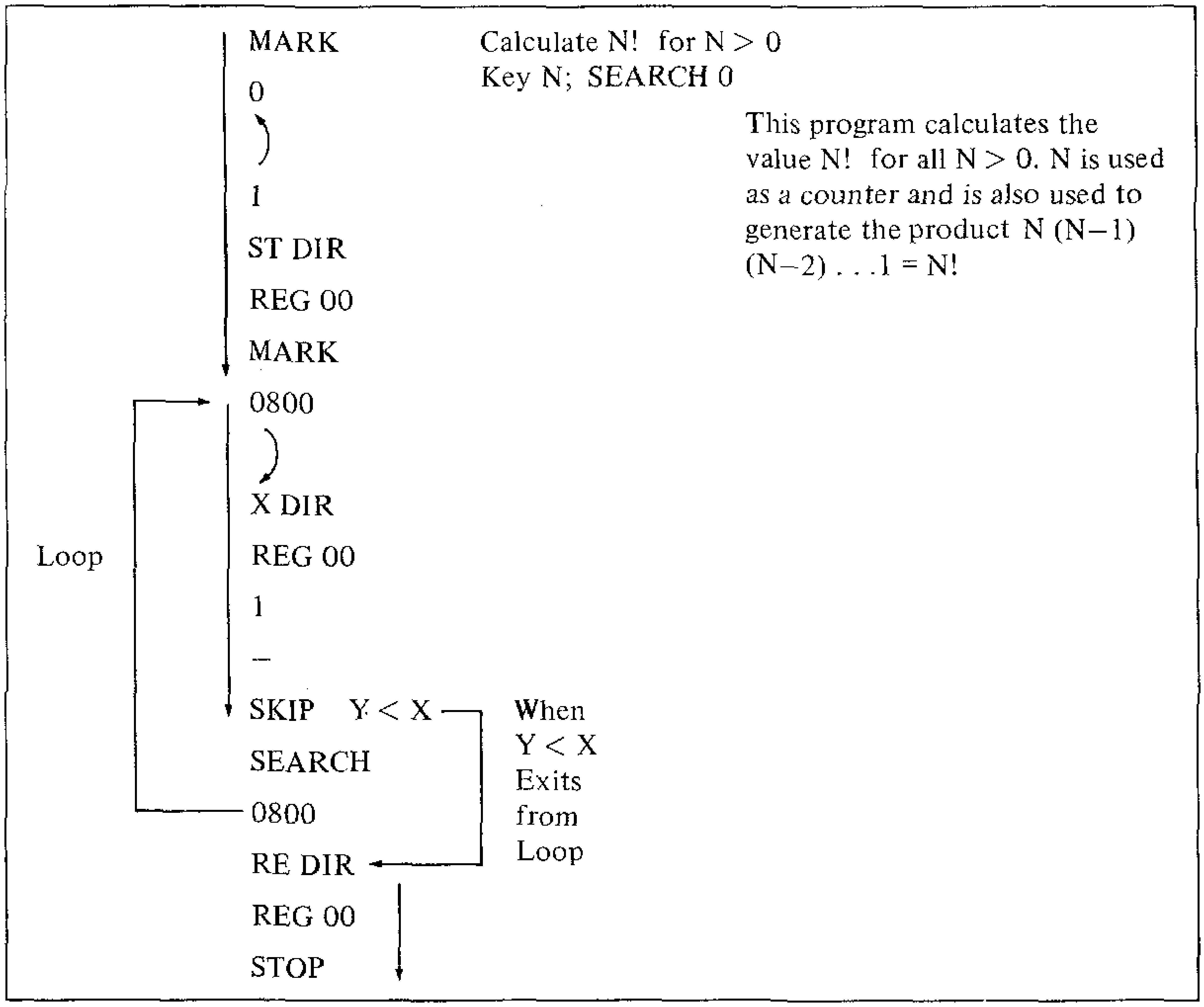
If Y < X the value $(Y + X)^2$ is calculated.



```
Path               MARK          Path        MARK
For Y ⩾ X          0             Y < X       0800
                   SKIP  Y ⩾ X               +
                   SEARCH                    )
                   0800                      x²
                   x²                        STOP
                   ( )
                   x²
                   +
                   )
                   √x
                   STOP
```

(3) Skip if Y < X

If the value in Y is less than the value in X, the program skips the next two steps. If the value in Y is equal to or greater than X, the next program step is executed.



```
         MARK        Calculate N!  for N > 0
         0           Key N; SEARCH 0
                                       This program calculates the
                                       value N!  for all N > 0. N is used
         1                             as a counter and is also used to
         ST DIR                        generate the product N (N-1)
         REG 00                        (N-2) . . .1 = N!
         MARK
         0800

         X DIR
Loop     REG 00
         1
         -
         SKIP   Y < X    When
         SEARCH          Y < X
         0800            Exits
         RE DIR          from
         REG 00          Loop
         STOP
```

(4) Skip if Error

The final decision command **SKIP IF ERROR** can be used in a variety of ways to check for certain conditions (see page 2-8 to review what operations turn the **PROGRAM ERROR INDICATOR** on). Testing for these conditions turns the program indicator off. The following program distinguishes between positive and negative numbers. If the number in X is positive, the program will branch to **MARK 0800.** If it is negative, the program finds the |X| and stops.

```
                MARK
                0
                √x
            ┌── SKIP IF ERROR  Path for x ≥ 0
     Path   │   SEARCH         Branch to MARK 0800
     for    │   0800           Note:  If x < 0 value in x
     x < 0  │                         remains unchanged,
            └─► |X|                   √x command simply
                STOP                  turns PROGRAM
                                      ERROR INDICATOR
                                      on.
```

The following program uses thc **SKIP IF ERROR** command to calculate the number of data registers a program occupies on the Wang 700. It also illustrates the **INTEGER X** command.

```
                 MARK              OPERATING INSTRUCTIONS
                 0                 1. Key number of program steps,
                 1                    SEARCH 0.
                 6                 2. Read number of data registers
                 ÷                    occupied, in Y.
                 ↓
                 INT X
                 −
                 ( )
                 1/x
If there is      SKIP IF ERROR ┐   Path if there is a remainder
a remainder    │ 1             │   of 0 after dividing by 16.
after divid-  {│ +             │   Division by 0 turns PROGRAM
ing by 16,     │ 2       ┌◄────┘   ERROR INDICATOR on.
the number     │ x       │         Indicator is turned off when
in Y must      │         │         SKIP IF ERROR command is
be increased   ↓ STOP    ↓         executed.
by 1.
```

PROGRAMMING TECHNIQUES

Looping Using a Counter

Looping is an important programming tool. The decision commands are most frequently used to set up loops within programs. Counters are set up to "count" the number of times a calculation is performed.

Suppose the sum Y = $x + x^2 + x^3 + ... + x^n$ is to be calculated for various values of x. The program below sets up a loop to calculate this sum for any number of terms. The value of n determines how many terms in the sum will be calculated.

```
MARK        | Key X
0           |
ST DIR      | Initializes
REG 00      | Registers
1           |
ST DIR      |
REG 01      |
0           |
ST DIR      |
REG 02      |
STOP        | Key n
↑           | counter in Y
MARK        |
0800        |
RE DIR      | x
REG 00      |
X DIR       |
REG 01      | x(x^(n-1))
RE DIR      |
REG 01      |
+ DIR       |
REG 02      | Σx^n
1           | Decrease N
-           | by 1
SKIP Y<X    |   Exits from
SEARCH      |   loop when
0800        |   N = 0
RE DIR      |
REG 00      |
RE DIR      |
REG 02      |
STOP        |
```

Loop is performed n times

Calculates x^n and Σx^n

OPERATING INSTRUCTIONS

Key x <u>SEARCH</u> <u>0</u>
Key N <u>GO</u>

Read x in Y-Register
Σx^n in X-Register

To understand the method used in the program, Rewrite the sum as $Y = T_0 + T_1 + T_2 + ... T_n$ where $T_0 = x$

$T_1 = xT_0$
$T_2 = xT_1$
$T_n = x\, T_{n-1}$

The program starts the sum with x and uses the recursive formula $x\,(x^{n-1})$ to calculate each successive term. N is used as a counter. The program performs the same calculation n times. Each time the loop is executed the counter is decreased by 1. When N = 0 the loop is terminated and the final sum is displayed.

Another slightly different counter is found in the program which stores 0's in the first 100 storage registers. In this example, the counter is constantly increasing until it reaches a value of 100. It also serves to designate the storage register being addressed.

```
MARK               PRIME
0                  SEARCH 0
0
ST INDIR
1
+
1
0
0
SKIP Y = X
SEARCH
0
STOP
```

Looping Without a Counter

Often there is no way to predetermine exactly the number of times a loop is to be performed. Other criteria have to be used.

EXAMPLE

In calculating the following sum for X > 1

$$1 + \frac{1}{X^2} + \frac{1}{X^4} + \frac{1}{X^6} + \frac{1}{X^8} + \ldots + \frac{1}{X^{2n}} ,$$

for specified accuracy the number of terms to be calculated depends on the value of x. However, it is obvious that each successive term gets smaller and smaller and eventually approaches zero. If 12 digits of accuracy are needed, the calculation can be carried out until the last term gets so small that it does not materially affect the overall sum. This occurs when the term becomes smaller than 10^{-11} and the overall sum on the 700 no longer changes its value when a term is added to it.

To write the program, it is convenient to rewrite the series as

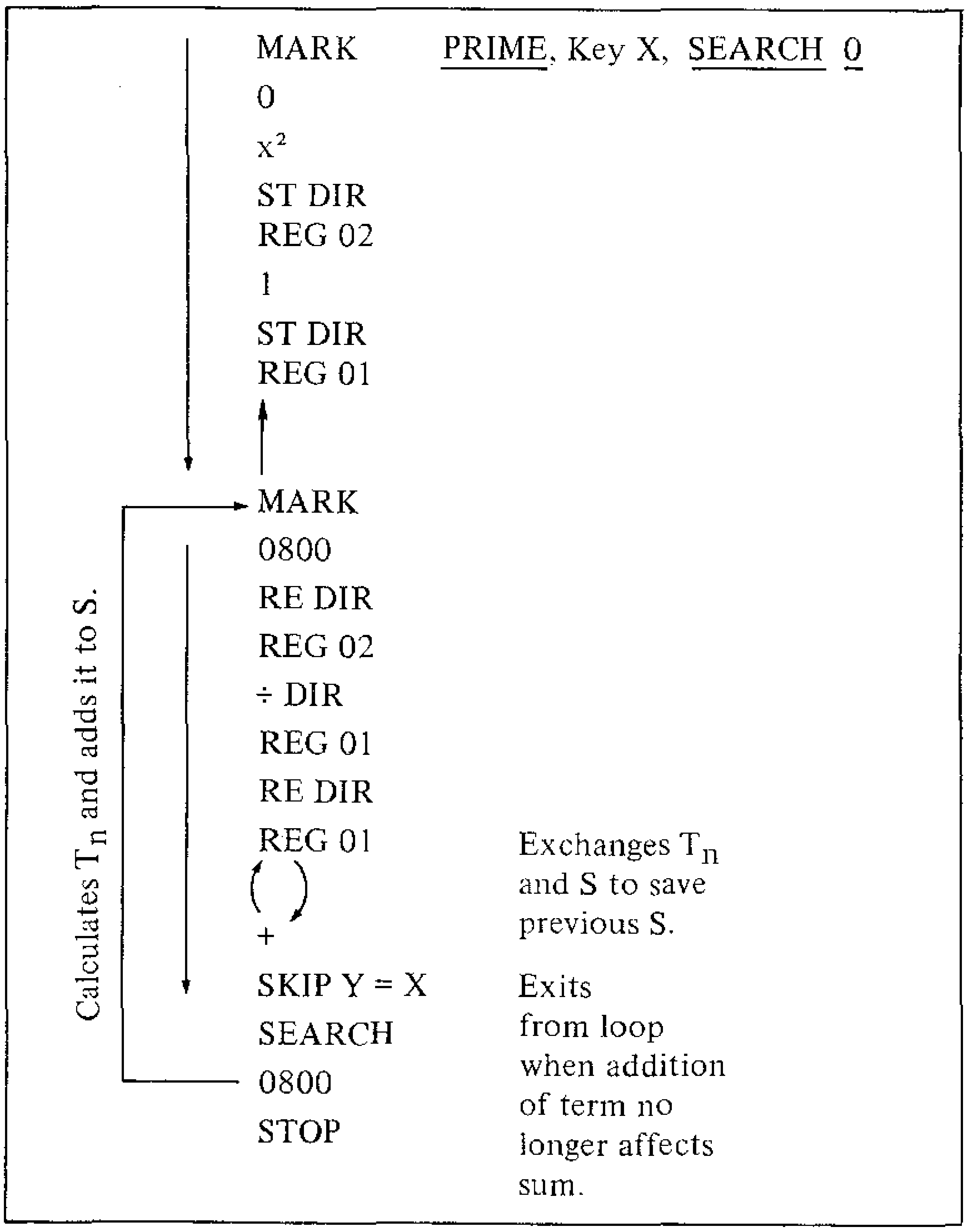
$$S = T_0 + T_1 + T_2 + \ldots + T_n \text{ where } T_0 = 1$$

$$T_1 = \frac{1}{x^2} T_0$$

$$T_2 = \frac{1}{x^2} T_1$$

.
.
.

$$T_n = \frac{1}{x^2} T_{n-1}$$



```
MARK        PRIME, Key X, SEARCH 0
0
x²
ST DIR
REG 02
1
ST DIR
REG 01
MARK
0800
RE DIR
REG 02
÷ DIR
REG 01
RE DIR
REG 01      Exchanges Tn
( )         and S to save
+           previous S.
SKIP Y = X  Exits
SEARCH      from loop
0800        when addition
STOP        of term no
            longer affects
            sum.
```

The program calculates the sum $\sum_{n=0}^{k} \frac{1}{x^n}$ to 12 significant digits

for any value of x. The loop is performed many more times for a smaller value of x than it would be for a larger value of x, simply because the series converges faster for large values of x. In all cases when $T_n < 10^{-11}$, the loop is terminated and the final sum is displayed in x and y. If only three digits of accuracy were needed, each successive term could be compared to 10^{-3}; and when $T_n < 10^{-3}$, the loop could be terminated.

Scanning a Table

Another frequent use for the decision command is to scan a table or schedule. In many situations, calculations or formulas vary with the class or range the input variable lies in.

A typical example is the pricing of articles. Discounts are often allowed according to the number of articles purchased. Below is a schedule for quantity discounts.

QUANTITY	DISCOUNT
0 to 10	0%
11 to 25	10%
26 to 50	13%
over 50	15%

When given the number of items to be purchased, this program calculates the discount figure which if multiplied by the unit price calculates total cost.

```
MARK                                   Key N = number of items to be
0                                              purchased SEARCH 0
↑
1                                      Read discount figure in Y, number
ST DIR                                 of items in X.
REG 00
1
1
SKIP Y ≥ X
SEARCH      BRANCH       →
0800        For N ≤ 10
.
1
– DIR
REG 00
2
6
SKIP Y ≥ X
SEARCH      BRANCH       →
0800        For 11 ≤ N ≤ 25
.
0
3
– DIR
REG 00
5
1
SKIP Y ≥ X
SEARCH      BRANCH
0800        For 26 ≤ N ≤ 50 →
```

```
.
0
2
– DIR         N > 50
REG 00
MARK
0800
RE DIR
REG 00
⇅
x
STOP
```

Go

The **GO** key is used to continue the program at the next step after the **STOP** instruction. One important technique that should be pointed out is the idea of using the **GO** command as a do nothing or no-operation instruction.

EXAMPLE

If two angles are unequal, we want to find the sine of the angle in X and use the sine of this angle in future calculations. If the two angles are equal, the angle itself will be used for future calculations. The program would be similar to the following:

STEP	KEY
000	MARK
001	0700
002	SKIP Y = X
003	SR 0002 (sinx)
004	GO
005	|X|
006	↑
007	π
008	x
009	.
	.

Executes subroutine if y ≠ x and returns to Step 004

Jumps to Step 005 if y = x

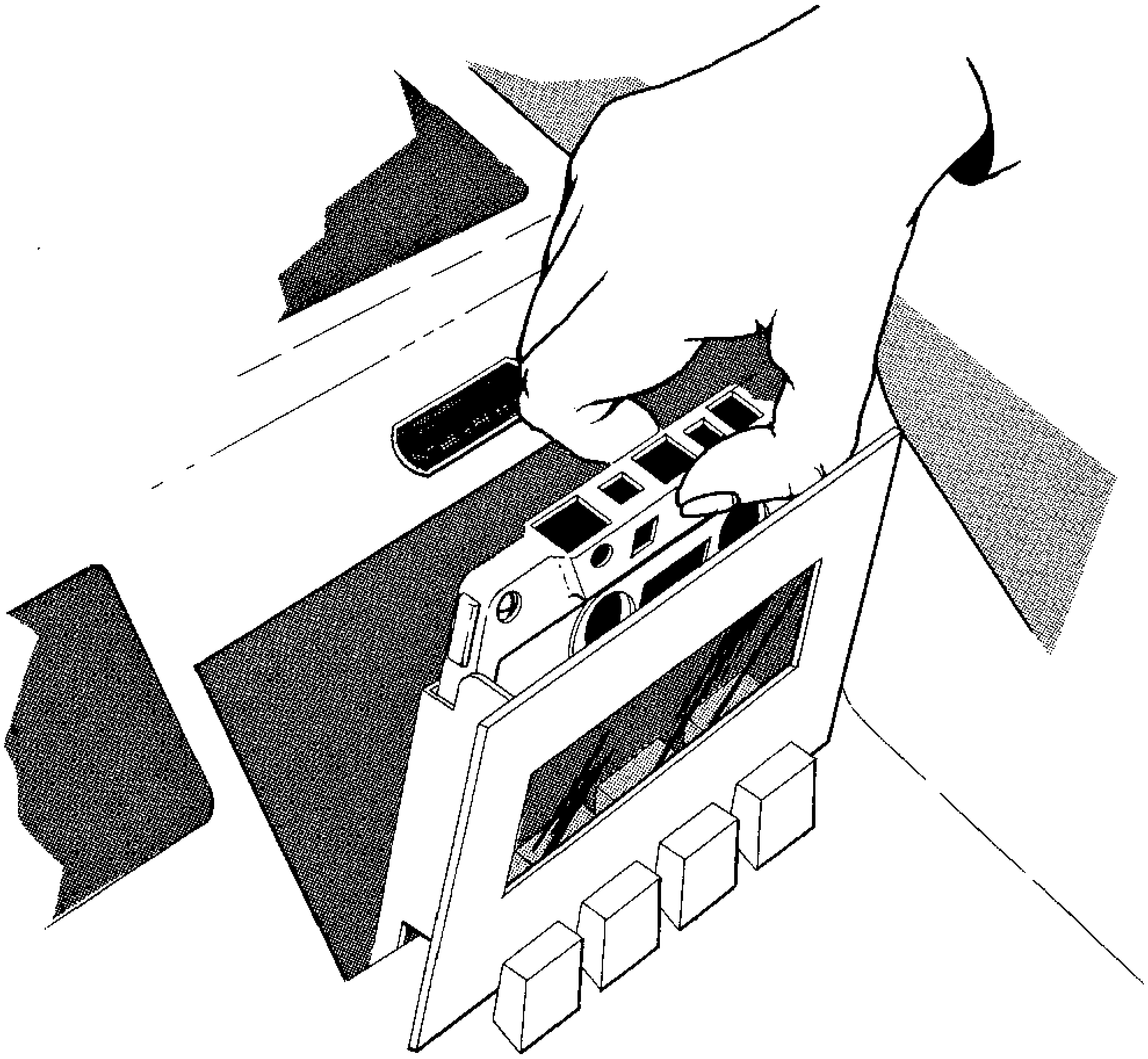
If the angles are unequal, the command SR 0002 tells the program to execute the sine subroutine. Upon completion of the subroutine, the program branches back to step 004. At this point, we do not wish to perform any operation because the **SKIP** command will skip two program steps if the condition is met. We want to perform the same calculation on the variable in X whether it is the sine of the angle or the angle itself. Therefore, a **GO** command is placed in step 004 which simply tells the program to continue on to the next step. In this way, the **GO** command can be used as a no-operation command which simply causes the program to continue on without changing or destroying any values.

SECTION VI PROGRAMMING TECHNIQUES USING TAPE CASSETTE

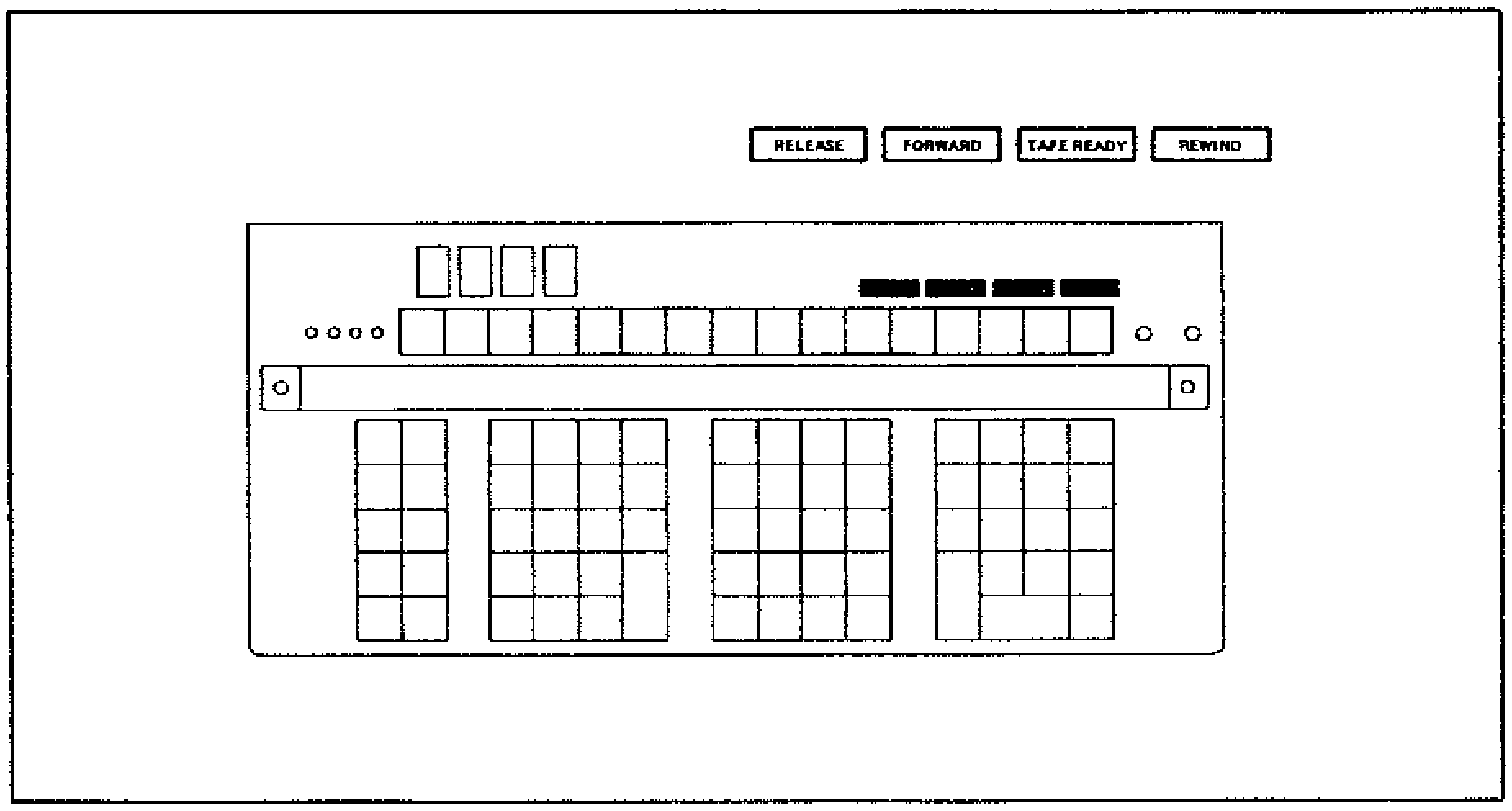
TAPE CASSETTE

Programs are saved on standard 4" x 2 1/2" x 1/2" magnetic tape cassettes for later use. Up to 20 blocks of programs can be saved on one tape cassette. The tape cassette consists of two tracks and each track can accommodate ten "program blocks."

INSERTING TAPE CASSETTE



TAPE DRIVE OPERATIONS



There are four basic buttons associated with the tape-drive mechanism.

1. The RELEASE button allows the operator to remove or insert his tape cassette.
2. The FORWARD button moves the tape in a forward direction when depressed.
3. The TAPE-READY button should be pushed when the 700 is to execute a tape instruction. This button places the head of the tape reader in contact with the tape.
4. The REWIND button rewinds the tape when depressed.

MACHINE ERROR INDICATOR

A MACHINE ERROR INDICATOR is located just to the right of the PROGRAM ERROR INDICATOR. If data is not transferred properly from or to the tape, the light will go on and the sign of the X register flashes. This indicates that the information has not been transferred properly and the operation should be repeated. This flashing light should NOT be confused with the PROGRAM ERROR INDICATOR located just to its left (See page 2-8). PRIME will turn both error indicators off.

HOW CAN A PROGRAM BE PROTECTED ONCE IT IS PUT ON TAPE?

There is no need to "erase" the tape. A new program will simply write over the old program. To insure that a good program stored on tape is not written over or lost accidently, each track of tape can be protected.

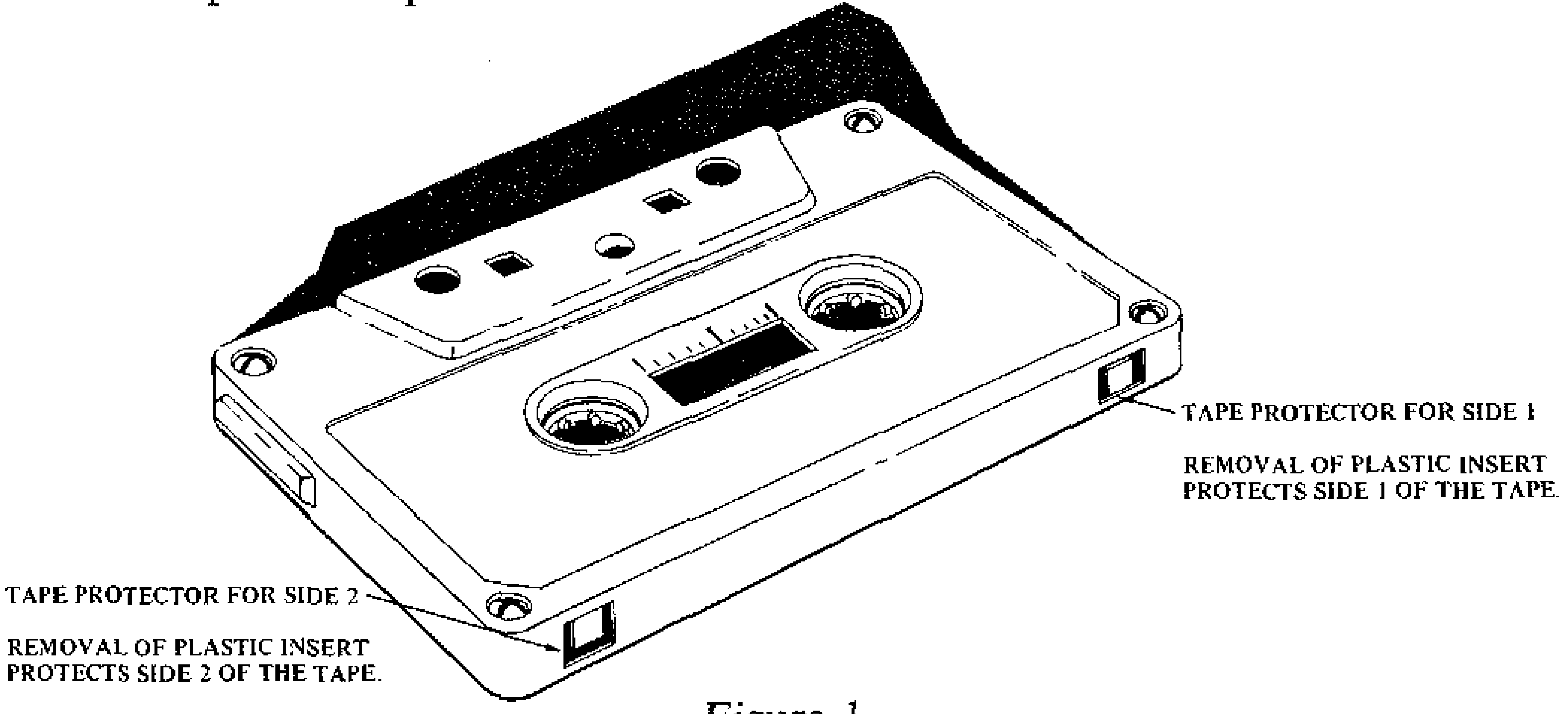

TAPE PROTECTOR FOR SIDE 1
REMOVAL OF PLASTIC INSERT PROTECTS SIDE 1 OF THE TAPE.
TAPE PROTECTOR FOR SIDE 2
REMOVAL OF PLASTIC INSERT PROTECTS SIDE 2 OF THE TAPE.

Figure 1

Figure I shows a top-view of the tape cassette. There are two small openings with a small plastic covering shown in Figure I. When this small plastic covering is removed, nothing can be recorded onto or erased from one side of the tape.

Once the plastic is removed, a piece of tape can be used to cover the opening if the tape is to be used for recording other programs.

WHAT IS A PROGRAM BLOCK?

A program block consists of any part of a program (up to 960 steps) which can be loaded into core at one time. It must be terminated by an **END PROGRAM** instruction. If an **END PROGRAM** instruction is not given and the **RECORD PROGRAM** key is indexed, the PROGRAM ERROR INDICATOR will go on after transferring all of core to the tape.

In this instance, the PROGRAM ERROR INDICATOR goes on because there is an error in programming (i.e., no END PROGRAM) and not a machine malfunction.

A program block must contain:

1. 960 program steps or less.
2. An **END PROGRAM** as a final instruction.

NOTE

If a program is 960 steps, the END PROGRAM is located at step 959. Even though the END PROGRAM command is not missing, the PROGRAM ERROR INDICATOR will go on when this program is transferred to tape. If this is the case (i.e., a 960 step program), simply PRIME and ignore the PROGRAM ERROR INDICATOR.

END PROGRAM

The **END PROGRAM** key defines a "program block." It is used to signal the end of a **RECORD PROGRAM** or **LOAD PROGRAM** operation. An **END PROGRAM** command is the last instruction to be transferred in a **RECORD PROGRAM** or **LOAD PROGRAM** instruction. Therefore, each program must be terminated by an **END PROGRAM** command if it is to be transferred onto tape.

It is recommended that only one **END PROGRAM** instruction be loaded into core at any one time. The primary reason for this is due to the **VERIFY PROGRAM** instruction. When the **VERIFY PROGRAM** key is indexed, the 700 always starts summing at step 000 and continues until an **END PROGRAM** instruction is encountered. Therefore, if additional programming instructions are located after the **END PROGRAM**, they will not be included in the sum generated by the **VERIFY PROGRAM** instruction. Therefore, when adding additional programming steps, write over the **END PROGRAM** instruction. This can be accomplished quite easily if it is remembered that the **PC** is set to the step that the **END PROGRAM** instruction occupies after a **VERIFY PROGRAM** has been executed. Therefore, after indexing **VERIFY PROGRAM,** simply index **LOAD PROGRAM** to load the additional steps or switch to **LEARN MODE** and start indexing them.

Always remember to end your program with an **END PROGRAM** instruction. This instruction is required for transferring the program from core to tape.

NOTE

An **END PROGRAM** *command must not be preceded by a program code whose high-order digit is 04. Logically, an instruction whose program code is 04XX would never precede an* **END PROGRAM** *command. (See code listing page 3-1.) One instance which might occur is the following:*

```
PROGRAM
•
•
•
MARK
0402
•
•
•
SEARCH
0402
END PROG
```

If the above program were loaded in core, the program would execute properly. However, if the program were to be transferred from core to tape the **END PROGRAM** *instruction would not be recognized as an* "**END PROGRAM**" *command. All of core would be transferred to tape and the* **PROGRAM ERROR INDICATOR** *would be turned on indicating there was no* **END PROGRAM** *terminating the program block.*

HOW TO "LEARN" A PROGRAM INTO CORE FROM THE KEYBOARD

A program is recorded into core by the following:

1. Place the 700 in LEARN MODE.
2. SET PC at the desired step where the first program command will be stored.
3. Index the program commands.

Remember to always end your program with an **END PROGRAM** instruction. This instruction is required for transferring the program from core to tape.

EXAMPLE

Program to find C: $C = \sqrt{a^2 + b^2}$ (Pythagorean Theorem)

STEP	KEY	CODE	
000	MARK	0408	Key a
001	0	0700	SEARCH 0
002	x^2	0713	Key b GO
003	↑	0604	Read c in X
004	STOP	0515	
005	x^2	0713	
006	+	0600	
007	↓	0605	
008	$\sqrt{x}$	0612	
009	STOP	0515	
010	END PROGRAM	0512	

The following instructions will introduce the above program into core:

1. Place 700 in LEARN MODE.
2. **SET PC** to the step you want the first program command to occupy. (To put the first program command at Step 000, **SET PC 0 0 0**.)

NOTE

An easy way to set the PC at 000 is by depressing the **PRIME** key.

3. Now simply index the program commands

MARK
0
x^2
↑
STOP
x^2
+
↓
$\sqrt{x}$
STOP
END PROGRAM

Notice while the program is introduced into core the PC displays the program step number and program code currently located at this step. Indexing a key causes the program code of the keystroke to replace the existent code. The PC is increased by one and displays the next step and current code. To see what is now loaded into core beginning at step 000, **PRIME** and step through your program.

KEYSTROKE	READ IN X		
PRIME	000	04	08
STEP	001	07	00
STEP	002	07	13

The above indicates that **MARK** is now stored at Step 000, **0** at Step 001, x^2 at 002 etc.

To execute the program, place the 700 in RUN MODE

Key a = 3 **SEARCH 0**
Key b = 4 **GO**
Read c = 5 in X

By stepping through the program in RUN MODE, each step will be executed one step at a time. In **LEARN MODE** the program is not executed. While stepping through the program in **RUN MODE**, one can see the step number and program code of the instruction about to be executed if the 700 is placed in **LEARN MODE**. Simply remember to put the 700 [illegible] in **RUN MODE** before indexing the **STEP** key; otherwise, the instruction will not be executed.

HOW TO TRANSFER A PROGRAM FROM CORE TO TAPE

RECORD PROGRAM

A program can be stored for later use on a magnetic tape.
To transfer a program from core to tape:

1. Place 700 in RUN MODE.
2. Insert the Tape cassette; push TAPE READY button.

3. **SET PC** to the first step of the program.
   (For the above example simply depress **PRIME** key.)
4. Index **RECORD PROGRAM** key, and all the steps from where the PC is set up to and including **END PROGRAM** will be loaded onto the tape. (For this example, Steps 000 to 010 are transferred to tape.)

NOTE

RECORDING DATA

Pairs of data storage registers can be recorded on magnetic tape for later use. See diagram (page 3-4). To transfer data from core to tape:

(1) Place 700 RUN MODE
(2) Insert Cassette, push TAPE READY
(3) SET PC to program step number corresponding to the data registers. An END PROGRAM command must be located immediately following the data.
(4) RECORD PROG

The same procedure to load the data in any pair of registers is used, except LOAD PROG replaces RECORD PROG in step (4).

HOW TO LOAD A PROGRAM FROM TAPE INTO CORE

The **LOAD PROGRAM** key transfers a program block on magnetic tape to core. The procedure is to set the PC and index the **LOAD PROGRAM** key. The first program command located on the tape is then stored in the step designated by the PC and the program continues loading until an **END PROGRAM** instruction is encountered. The **END PROGRAM** instruction is the last step to be loaded into core. When the loading is terminated, the PC is automatically set to the first step just loaded into core.

In general, four steps are required for loading a program:

1. Place 700 in RUN MODE.
2. Insert Tape Cassette, push TAPE READY button.
3. **SET PC** to first step in core that program will occupy.
4. Index **LOAD PROGRAM.**

Since ten program blocks can be saved on one side of a tape cassette, what happens if the third block is desired?

If the third block is to be loaded beginning at Step 000:

1. Place 700 in RUN MODE.
2. Insert tape cartridge, REWIND tape completely, push TAPE READY button.
3. **PRIME** (Sets PC to Step 000).
4. Index **LOAD PROGRAM** (loads 1st block and sets PC to 000).
   LOAD PROGRAM (loads 2nd block and sets PC to 000).
   LOAD PROGRAM (loads 3rd block and sets PC to 000).

The third block on the tape is now loaded into core and is ready to be executed. Normally, the sequence of steps to follow in loading the nth block of the tape into core is to index the **LOAD PROGRAM** key "n".times. As a final check, the **VERIFY PROGRAM** key can be indexed and the verify program number checked to be completely sure the correct program is loaded. The **VERIFY PROGRAM** key starts summing from Step 000 until it encounters an **END PROGRAM** instruction.

BYPASSING PROGRAM BLOCKS

In some instances, loading the preliminary program blocks on a tape will destroy parts of core which must be saved. The problem arises as to how to bypass a program block without loading it into core and destroying data which will be needed for later calculations.

If the PC is set to 944 and the **LOAD PROGRAM** key indexed, the program will only load into Steps 944 to 959. If the program is greater than 16 steps, it will continue loading within those steps (944 to 959). In other words, when a program exceeds Step 959 in core, the remaining part of the program is simply loaded over itself in the first two data registers 000 and 001. The **PROGRAM ERROR INDICATOR** will go on when this occurs.

Therefore, by destroying only the contents of Registers 000 and 001, the first three blocks of tape can be bypassed.

1. Load tape and place 700 in RUN MODE:
2. **SET PC 9 4 4.**
3. **LOAD PROGRAM** (bypasses 1st block)
   LOAD PROGRAM (bypasses 2nd block)
   LOAD PROGRAM (bypasses 3rd block).
4. **PRIME** (to turn off PROGRAM ERROR INDICATOR).
5. **SET PC** to first step that the desired program will occupy.

> NOTE
> *If the program is to be loaded beginning at Step 000, PRIME will have already set the PC to 000.*

The fourth block of tape is now stored in core and only registers 000 and 001 have been altered.

PROCEDURE FOR CORRECTING SINGLE PROGRAM STEPS

It is quite easy to correct any part of a 700 program. Suppose the following program to accumulate the sum Σx in Register 28 and Σx^2 in Register 29 is loaded into core as shown.

STEP	KEY	CODE
020	MARK	0408
021	1	0701
022	STOP	0515
023	– DIR	0401
024	REG 28	0208
025	x^2	0713
026	+ DIR	0400
027	REG 29	0209
028	SEARCH	0407
029	1	0701
030	END PROGRAM	0512

Notice at Step 023 the – DIR key has been entered by mistake. Correcting this error requires three steps:

1. Place 700 in LEARN MODE.
2. Set PC at step to be corrected:
   SET PC 0 2 3
3. Index correct key:
   + DIR

The correct step + DIR is now located at Step 023. In a similar way, any step in core can be directly assessed using the SET PC key and the correct step keyed in.

PROCEDURE FOR INSERTING EXTRA PROGRAM STEPS

Suppose a 400 step program has been introduced into core and three steps which occur in the middle of the program have been omitted. These steps can be inserted without having to key in the entire program again. Using the same program as on the previous page, suppose the two steps 1 + should appear between Steps 021 and 022 to indicate which x is about to be entered. The procedure for inserting the steps is as follows:

1. Insert a tape cartridge, REWIND tape completely, and push TAPE READY.
2. **SET PC 0 2 2 RECORD PROGRAM.** (This instruction saves the second half of the program, Steps 022 to 030, by transferring these steps to tape. The PC is set to Step 022 when the instruction is completed.)
3. Be sure 700 is in LEARN MODE and index the steps to be inserted:
   1 (Loaded at Step 022, PC increases to 023.)
   + (Loaded at Step 023, PC increases to 024.)

   (Notice that the added steps are loaded into core at the proper place because the PC is set to the first instruction transferred to tape by the **RECORD PROGRAM** instruction.)
4. Put 700 in RUN MODE, REWIND tape, and push TAPE READY.
5. LOAD PROGRAM (The steps saved on the tape are now loaded into core beginning at Step 024.)

The extra steps have been inserted and the program is ready to be executed. The program appears in core as illustrated:

STEP	KEY	CODE
020	MARK	0408
021	1	0701
022	1	0701
023	+	0600
024	STOP	0515
025	+ DIR	0400
026	REG 28	0208
027	x^2	0713
028	+ DIR	0400
029	REG 29	0209
030	SEARCH	0407
031	1	0701
032	END PROGRAM	0512

PROGRAMMING TECHNIQUES USING TAPE CASSETTE

An interesting feature of the **LOAD PROGRAM** key is that it is programmable. This allows different parts of a program to use the same place in memory at different times. When a **LOAD PROGRAM** instruction is encountered in a program, the next program block is loaded into core beginning at the step immediately following the **LOAD PROGRAM** command. Immediately after the **END PROGRAM** instruction is loaded, control switches to the first instruction loaded and the program is executed.

To take advantage of this command a loop has to be formed. An example illustrates this idea.

EXAMPLE

```
      PROGRAM IN CORE                   PROGRAM ON TAPE

   Step #      COMMAND                  2
    000       MARK                      ↑
    001       0                         +
    002       LOAD PROG                 ↓
    003       END PROG                  ST DIR
    004                                 REG 00
    005                                 SEARCH
    006                                 0
                                        END PROG
                                      ===========
                                        3
                                        ↑
                                        +
                                        ↓
                                        + DIR
                                        REG 00
                                        STOP
                                        END PROG
```

This program will illustrate how to use the **LOAD PROGRAM** instruction in programming and also points out the importance and need for the **END PROGRAM** instruction. The program consists of three parts or blocks. The first part is loaded into core and consists of four instructions.

The program is started by a **SEARCH 0** from the keyboard. The **LOAD PROGRAM** is the first instruction. The 700 immediately starts loading the next program block into core. After the **LOAD PROGRAM** instruction is executed, core will look like the following:

STEP	COMMAND
000	MARK
001	0
002	LOAD PROG
003	2
004	↑
005	+
006	↓
007	ST DIR
008	REG 00

009	SEARCH
010	0
011	END PROG

The program will immediately add 2 + 2 and store the sum in Register 000. The program then encounters a **SEARCH 0**. Control switches back to **MARK 0** and the **LOAD PROGRAM** instruction is encountered causing the next block to be loaded in replacing the last block. *Notice the* **END PROGRAM** *instruction was never used in executing the* program: however, it was needed when the **LOAD PROGRAM** instruction was first executed. If an **END PROGRAM** command had not been located after the **SEARCH 0** command, the 700 would have continued to load programming steps into core. The **END PROGRAM** instruction tells it where to stop.

STEP	COMMAND
000	MARK
001	0
002	LOAD PROG
003	3
004	↑
005	+
006	↓
007	+ DIR
008	REG 00
009	STOP
010	END PROG

The program adds 3 + 3 and adds the sum to Register 000 and stops.

The program is only used to demonstrate how to program the **LOAD PROGRAM** *instruction*. This technique of a **LOAD PROGRAM** within a program should only be used in long programs which require many program steps and data registers. A valid example might be in a multiple regression analysis where registers are needed for storing sums. In addition, a program for generating the sums and solving simultaneous equations is needed. Therefore, *the first program block could initialize all registers and generate the numerous sums needed.* When the routine was finished, the second block to solve the simultaneous equations and find the coefficients could be called and loaded into the same part of memory that the first block occupied. In this way, memory can be shared and utilized to its fullest extent.

CREATING A MULTI-BLOCK TAPE

The idea of sharing core storage presents the problem of creating a multi-block tape. The simplest way to explain this procedure is by creating the 3-block program.

I. To create the first block:

A. Key the first program block into core.

1. Set **LEARN** mode
2. **PRIME** (more generally SET PC to location of first program step)
3. **MARK**
   0
   LOAD PROG
   END PROG.

B. Transfer this block to tape

1. Insert tape cassette, **REWIND**
2. Set **TAPE READY**
3. **PRIME** (or **SET PC** to first step)
4. **RECORD PROG**

II. To create the second block:

A. Key second program block into core

1. **PRIME** (or **SET PC** to first step)
2. 2
   ↑
   +
   ↓
   ST DIRECT
   00
   SEARCH
   0
   END PROG

B. Transfer this block to tape

1. **PRIME** (or **SET PC** to first step)
2. **RECORD PROG**

III. To create the third block:

A. Key third program block into core

1. **PRIME** (or **SET PC** to first step)
2. **3**
   ↑
   +
   ↓
   + DIR
   00 (Toggles down)
   STOP
   END PROG

B. Transfer block to tape

1. **PRIME** (or **SET PC** to first step)
2. **RECORD PROG**

To run the program:

1. **PRIME**
2. **REWIND** the tape to beginning
3. Set **TAPE READY**
4. **LOAD PROG**
5. **SEARCH 0**

The program will stop with a 6 in X and Y. To recall the sum, **RECALL DIRECT 00** and 10 will appear in X.

NOTE

The tape is not moved manually while creating the multi-block tape. If an entirely unrelated program were to be added to this tape as the fourth block, initializing the tape would consist of bypassing the first three program block as discussed in Section VI, page 6-8.

EXAMPLE

Suppose a program for calculating the t-test for paired variates were located in core from step 100 to 155 and it is to be recorded as the fourth block on the multi-block tape just created.

1. Insert tape cassette, **REWIND**
2. Set **TAPE READY**
3. Set **RUN** mode
4. **SET PC** 9 4 4
5. **LOAD PROG** (bypasses first three program blocks)
   LOAD PROG
   LOAD PROG
6. **PRIME** (to turn PROGRAM ERROR INDICATOR off)
7. **SET PC** 1 0 0
8. **RECORD PROG**

The t-test for paired variates is now recorded as the fourth block on this tape.

SECTION VII ADDITIONAL COMMANDS NOT FOUND ON THE 700 KEYBOARD

PAUSE COMMAND

One remaining function not previously discussed is the 700 pause command.

WRITE A PAUSE

The WANG 700 has a pause command which allows the user to display the X and Y Registers for .5 seconds at any predetermined point within a program.

The command is a two-step instruction. Since it should only be used within a program, it has not been assigned a regular key on the 700 keyboard.

The two-step command is:

WRITE A followed by the code 0615 which corresponds to the 1/x key.

The following program will count from 0 to 10 displaying each integer in Y for .5 seconds.

KEY	CODE	COMMENT	OPERATING INSTRUCTIONS
MARK	0408		SEARCH O
0	0700		
1	0701		
+	0600		
WRITE A	0412	2-Step command	
1/x	0615	causes .5 second	
1	0701	pause	
0	0700		
SKIP IF Y = X	0509		
SEARCH	0407		
0	0700		
STOP	0515		

Multiple PAUSE commands can be used if a longer pause is required. Simply repeat the two-step command for each half second pause.

The pause command is only operational under programming mode. It cannot be used to cause a stop or pause in a program that is executing. If a program is executing and the user desires to stop it at any point, simply index the **STEP** key and the program will stop instantly (See Section II, page 2-3).

In addition to the various commands found on the 700 keyboard, there are several powerful commands which have not been assigned special keys on the Wang 700. These commands are used primarily in programming applications. One of these commands, PAUSE, was discussed above.

These special programming commands can be divided into three basic categories:

1. Storage commands
2. Decision commands
3. Shifting commands

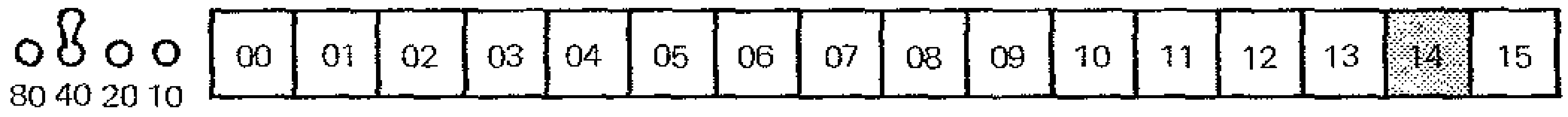
STORAGE COMMANDS (DIRECT ACCESS TO AND FROM THE Y-REGISTER)

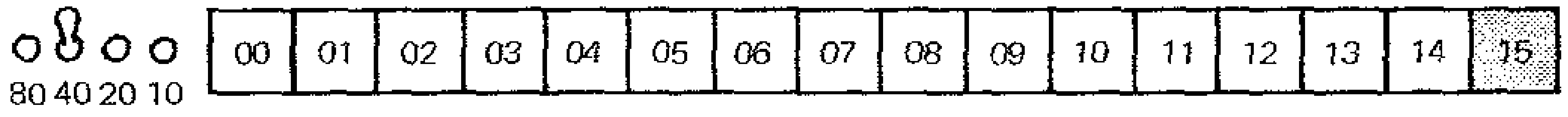
It is possible to transfer data directly to and from the Y-Register and any of the 122 data storage registers. The two-step command is similar to the **RECALL DIRECT** and **STORE DIRECT** commands except that the Y-Register is used in the data transfer rather than the X-Register. The first instruction specifies whether to store or recall, the second instruction designates the internal data register.

STORE Y	0414	Stores the number in Y into the data register designated by the next keystroke. Y unchanged, X unchanged.
DATA REGISTER	XXXX	
RECALL Y	0415	Recalls to Y the number from the register designated by the next keystroke. Designated register unchanged, X unchanged.
DATA REGISTER	XXXX	

Since the results of the arithmetic operations using X and Y are placed in Y, the **STORE Y** command saves the program step (↓) of moving the number down into the X-Register when the result is to be saved.

> **NOTE**
>
> *The program codes 0414 and 0415 must be entered by using the toggle switches and special function keys.*





DECISIONS

In addition to the four decisions available from the keyboard (See Section V, page 5-1), there are eight other conditions which can be tested. These commands require two instructions. They are each preceded by a **WRITE ALPHA** command and use an existing key on the 700 keyboard for the second half of the command. They are listed as follows. They test for a positive, negative, zero, and non-zero value in the X and Y Registers.

X-REGISTER

WRITE ALPHA SET EXP	Skips next 2 instructions if X is negative
WRITE ALPHA $LOG_{10} X$	Skips next 2 instructions if X is positive
WRITE ALPHA CHANGE SIGN	Skips next 2 instructions if X is not zero
WRITE ALPHA $LOG_e X$	Skips next 2 instructions if X is zero

Y-REGISTER

WRITE ALPHA SKIP IF ERROR	Skips next 2 instructions if Y is negative
WRITE ALPHA GROUP II	Skips next 2 instructions if Y is positive
WRITE ALPHA RETURN	Skips next 2 instructions if Y is not zero
WRITE ALPHA WRITE	Skips next 2 instructions if Y is zero

If the condition is met, the next 2 programming instructions are skipped. If the condition is not met, the program continues with the next step.

EXAMPLE:

The following program calculates 2^n *and illustrates the two-step command* **WRITE ALPHA, WRITE** which checks for Y equal to zero.

OPERATING PROCEDURE
Key N; **SEARCH 0**
Read 2^N in X

```
000   MARK
001   0           Initializes
002   ↑           Register
003   1           N in Y
004   STORE DIR
005   REG 00
006   MARK        Inner Loop
007   0800        Calculates
008   2           2^N
009   X DIR                           017   RE DIR
010   REG 00                          018   REG 00      RECALLS ANSWER
011   1                               019   STOP        TO X
012   –
013   WRITE A     2 Step Command
014   WRITE       SKIP IF Y = X
015   SEARCH      Exits When
016   0800        Y = 0
```

SHIFTING COMMANDS:

The following two-step commands give the user an easy way to shift the decimal point of the X-Register from 1 to 10 places, left or right.

The first group shifts the decimal point n-places to the right and effectively multiplies the X-Register by 10^n.

WRITE ALPHA 1	Multiplies X by 10^1
WRITE ALPHA 2	Multiplies X by 10^2
WRITE ALPHA 3	Multiplies X by 10^3
WRITE ALPHA 4	Multiplies X by 10^4
WRITE ALPHA 5	Multiplies X by 10^5
WRITE ALPHA 6	Multiplies X by 10^6
WRITE ALPHA 7	Multiplies X by 10^7
WRITE ALPHA 8	Multiplies X by 10^8
WRITE ALPHA 9	Multiplies X by 10^9
WRITE ALPHA 0	Multiplies X by 10^{10}

The second group shifts the decimal point n-places to the left and effectively divides the X-Register by 10^n.

WRITE ALPHA – DIRECT	Divides X by 10^1
WRITE ALPHA X DIRECT	Divides X by 10^2
WRITE ALPHA ÷ DIRECT	Divides X by 10^3
WRITE ALPHA STORE DIRECT	Divides X by 10^4
WRITE ALPHA RECALL DIRECT	Divides X by 10^5
WRITE ALPHA EXCHANGE DIRECT	Divides X by 10^6

WRITE ALPHA SEARCH	Divides X by 10^7
WRITE ALPHA MARK	Divides X by 10^8
WRITE ALPHA GROUP I	Divides X by 10^9
WRITE ALPHA + DIRECT	Divides X by 10^{10}

EXAMPLE

If X contains **12.3456781245** and the command **WRITE ALPHA 3** is given X will then contain **12345.6781245.**

If the command **WRITE ALPHA SEARCH** is given, X will contain **.123456781245-02**

These commands are extremely useful in applications where scaling of input and/or output must be accomplished.

SECTION VIII TRIGONOMETRIC AND STATISTICAL PACKAGE PROGRAMS

THE TRIG PACK

The TRIG PACK on the WANG 700 consists of the following 16 trigonometric functions:

SPECIAL OPERATION KEY	TRIG FUNCTION	INPUT RANGE
00	DEGREES TO RADIANS	$\|x\| < 10^{99}$
01	RADIANS TO DEGREES	$\|x\| < 10^{98}$
02	SINE X	$\|x\| < 10^{99}$
03	COSINE X	$\|x\| < 10^{99}$
04	TANGENT X	$\|x\| < 10^{99}$
05	SIN^{-1} X	$\|x\| \leqslant 1$
06	COS^{-1} X	$\|x\| \leqslant 1$
07	TAN^{-1} X	$\|x\| < 10^{99}$
08	TO POLAR	$\|x\| < 10^{50}; \|y\| < 10^{50}, \neq 0$
09	TO RECTANGULAR	$0 \leqslant R < 10^{99}$ $\|\theta\| < 10^{99}$
10	SINHX	$\|x\| < 227.9$
11	COSHX	$\|x\| < 227.9$
12	TANHX	$\|x\| < 227.9$
13	$SINH^{-1}$ X	$-10^{7} < x < 10^{50}$
14	$COSH^{-1}$ X	$\|x\| \geqslant 1$
15	$TANH^{-1}$ X	$\|x\| < 1$

These functions are loaded into core memory from a tape cassette which is provided with the machine. The standard TRIG PACK consists of 384 program steps (or 48 Data Registers).

NOTE

The TRIG PACK also uses five data registers:700A are 000, 001, 002, 120 and 121, 700B are 000, 001, 002, 003 and 120 registers. Care should be taken in using these registers — information stored in these five registers would be lost after an execution of one of the trig functions.

The Y-Register is always preserved and remains unchanged except in the POLAR and RECTANGULAR conversions. When the entire TRIG PACK is loaded into core, core storage for the 700A is as follows:

CORE MEMORY USAGE BY THE TRIG PACK

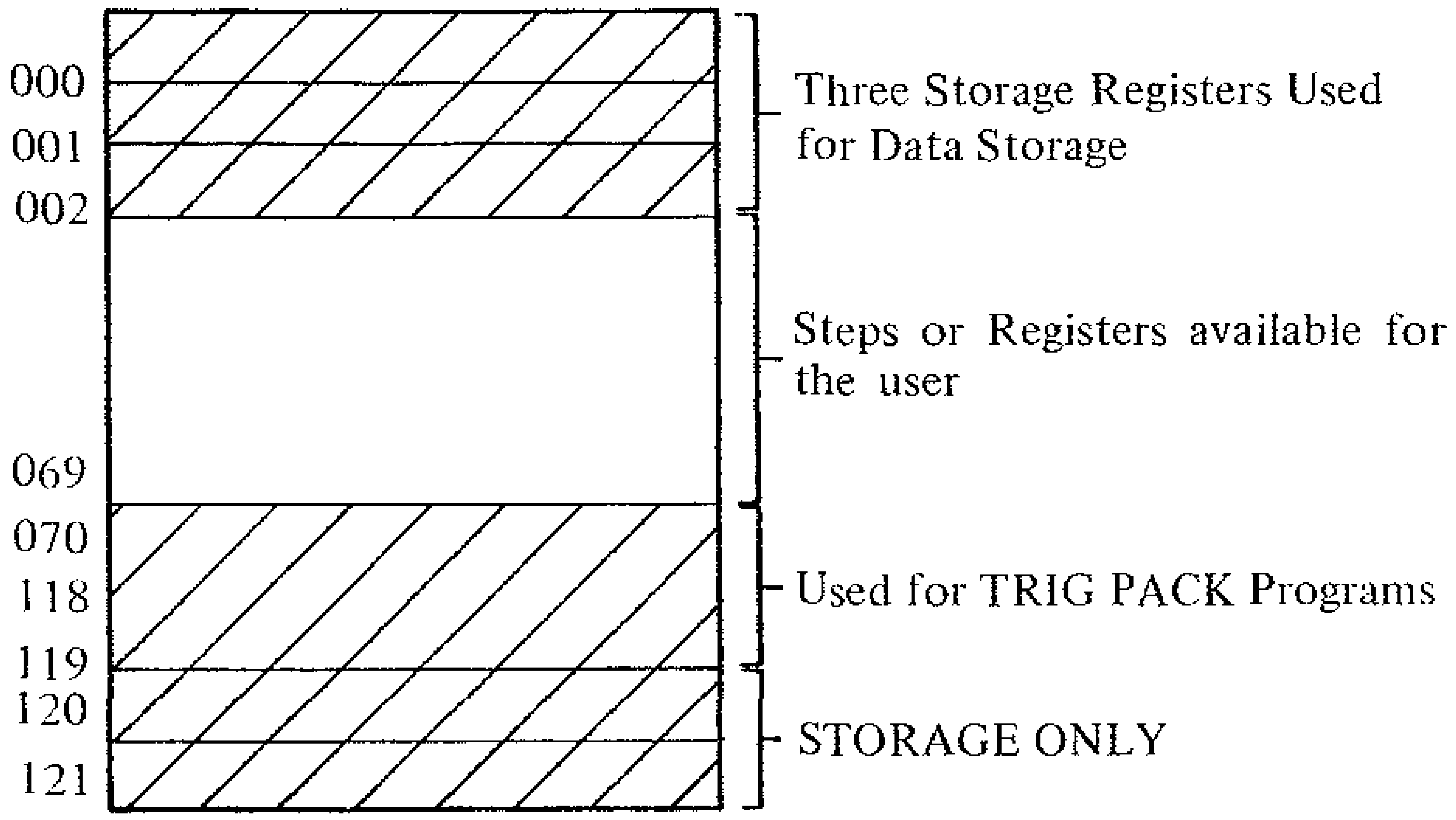

000
001
002
069
070
118
119
120
121
Three Storage Registers Used for Data Storage
Steps or Registers available for the user
Used for TRIG PACK Programs
STORAGE ONLY

FIGURE 1

SPEED AND ACCURACY

The speed for each function varies. In the worst case, it is no longer than 250 milliseconds. Accuracy is 10 significant digits.

TO LOAD THE TRIG PACKAGE

Like any other tape-to-core operation, the TRIG PACK is loaded as follows:

1. Insert trig-tape, **REWIND**
2. Set **RUN**
3. Set **TAPE READY**
4. **PRIME, LOAD PROGRAM**

By following these steps, the TRIG PACK is loaded into core starting at Step 000 and utilizes core as indicated in Figure 1. It is recommended that the TRIG PACK always be loaded starting at Step 000.

In addition to the TRIG PACK, most users will want to load their own programs into core. In order not to erase any of the TRIG PACK, other programs should be introduced into core **beyond** the TRIG PACK. The **VERIFY PROGRAM** key allows us to bypass the TRIG PACK quite easily. After depressing the **VERIFY PROGRAM** key, the PC is set to the step the **END PROGRAM** instruction occupies.

Since it is advisable to have only one **END PROGRAM** instruction in core at any one time, additional programs should start at this step where the **END PROGRAM** instruction is located.

TWO CASES exist:

1. Indexing additional programming steps from the keyboard.
2. Loading another program into core from another tape.

In each case the procedure is basically the same.

CASE 1: Adding Steps from the Keyboard.

1. After loading the TRIG PACK, place 700 in LEARN MODE
2. **VERIFY PROGRAM.**
3. Key program steps desired.

The first step indexed, most likely a **MARK**, will replace the **END PROGRAM** command of the TRIG PACK. Therefore, after completing your own program, an **END PROGRAM** command has to be given. This will complete a new block consisting of the TRIG PACK plus your own program. The **VERIFY PROGRAM** number will then total the codes of the TRIG PACK and your own program.

CASE 2: Loading Another Program into Core from Tape

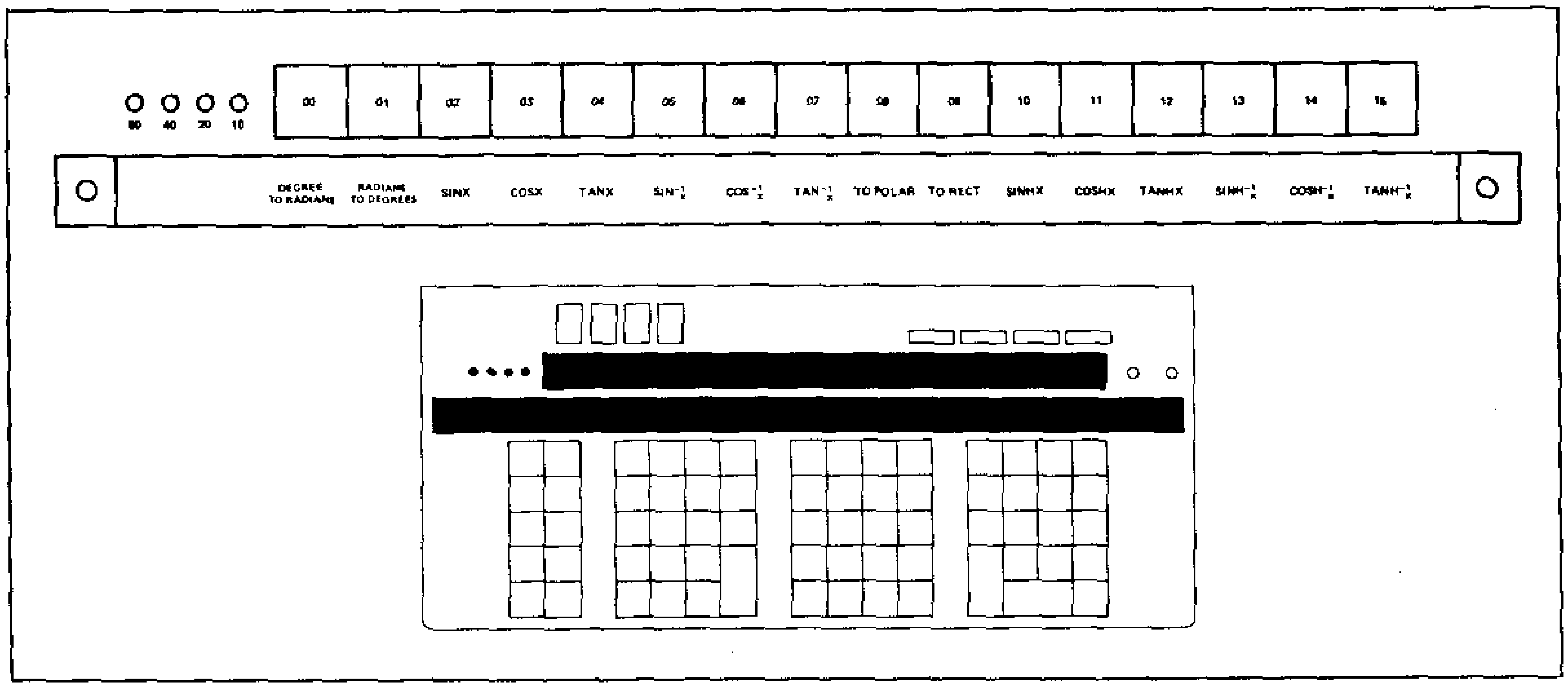
1. After loading the TRIG PACK, insert TAPE CASSETTE which has desired program .
2. Leave 700 in RUN MODE, Set **TAPE READY**
3. **VERIFY PROGRAM** (bypasses TRIG PACK).
4. **LOAD PROGRAM .**

The program will be loaded into core directly following the TRIG PACK.

USING THE TRIG PACKAGE

The TRIG PACK consists of 16 subroutines which can be addressed from the keyboard or under program control (See page 8-6).

KEYBOARD USE: BE SURE THE TOGGLE SWITCHES ARE IN THE OFF (DOWN) POSITION WHEN ADDRESSING THE TRIG FUNCTIONS FROM THE KEYBOARD.



PROGRAM USE

It should be clear that the TRIG PACK is using the special function keys to address the first 16 subroutine codes discussed in Section IV, page 4-5. For instance, the SINE routine is prefixed by a **MARK 0002** and terminated by a **RETURN** command. If the subroutine is addressed within a program, the **RETURN** command transfers control back to the main program. If the subroutine is addressed from the keyboard, control is transferred back to the keyboard. The following shows a user's program utilizing the SINE and COSINE routines.

EXAMPLE: Find Y = 2 sinθ cosθ

KEY	CODE		
MARK	0408		OPERATING INSTRUCTIONS Key θ in Degrees **SEARCH 0** Read 2 cosθ sin θ in Y
0	0700		
↑	0604	STORES θ in Y	
SINX	0002	FINDS SINE θ	
↓↑	0606	SINE θ in Y θ in X	
COSX	0003	FINDS COS θ	
X	0602	Y = sinθ cos θ	
2	0702		
X	0602	Y = 2 sinθ cos θ	
STOP	0515		

The program makes use of the fact that the Y-Register is preserved by storing θ in Y and then the SIN θ in Y.

DESIGN OF THE TRIG PACKAGE

The TRIG PACK has been designed to give the user greater flexibility. Since the TRIG PACK resides in core memory and is not a "hardware" feature, certain functions which are not used often can be easily deleted. For example, a user may only need SINX, COSX, and TAN^{-1} X for his calculations.

By setting the PC to the step number following these functions, the rest of the TRIG PACK can be deleted and more core storage for other programs and data storage can be gained.

STATISTICAL PACKAGE PROGRAM

In the same way, statistical users will load into core the STATISTICS PACK rather than the TRIG PACK. The STATISTICS PACK will consist of the following functions and will be loaded into core in the same way as the TRIG PACK.

FUNCTION	KEY
Mean, Variance, Standard Deviation (ungrouped)	00
Mean, Variance, Standard Deviation (grouped)	01
Normal Distribution	02
Inverse Normal Distribution	03
X^2 Statistic	04
X^2 Distribution	05
Error Function	06
Binomial Distribution	07
N!	08
Linear Regression	09
Gamma Function	10
Negative Binomial Distribution	11
Poisson Distribution	12
Random Number Generator	13

ASSIGNMENT OF SPECIAL OPERATION KEYS FOR A USER'S OWN SUBROUTINES

The concept of a subroutine was discussed briefly in Section IV, page 4-5. Sixty-four codes are reserved for subroutines on the WANG 700. A subroutine is addressed by a **single** keystroke or a **single program** step. It is prefixed by a **MARK XXXX** chosen from the 64 reserved codes and terminated by a **RETURN** command. The TRIG and STATISTICS packages use the first 16 subroutine codes listed on page 4-5, and are easily addressed by the SPECIAL FUNCTION keys when the Toggle Switches are all in the OFF (DOWN) position.

If Toggle Switch 10 were placed in the ON (UP) position, indexing the **03** key would cause the 700 to look through core for the subroutine beginning with **MARK 0103.** Remember there are 64 codes which can be used as subroutines – not merely 16.

The SPECIAL FUNCTION keys can be used to address the user's own custom-made functions rather than those found in the TRIG and STATISTICS PACKS. Any subroutine which requires only one piece of input data can be addressed by any of the SPECIAL FUNCTION keys.

While in LEARN MODE, the user simply presses **MARK** followed by one of the SPECIAL FUNCTION keys. This will set the **MARK** flag in core for direct access to the assigned routine. At the end of the subroutine a **RETURN** is given.

For example, assigning 12 to y = πr^2

KEY	CODE
MARK	0408
12	0012
x^2	0713
↑	0604
π	0609
x	0602
RETURN	0511

To call for this function simply index r into X, and press 12. The answer will be given in Y. The subroutine can also be addressed under program control in the same way as the TRIG functions. In this way the user may assign and label any of his own functions to the special operation keys.

SECTION IX
SAMPLE PROGRAMS

700 PROGRAM DESCRIPTION

PROGRAM TITLE ALGEBRA OF COMPLEX NUMBERS +, -, x, ÷	NUMBER 1004A/MA6
PROGRAM ABSTRACT +, -, x, and ÷ complex numbers	PROGRAMMED BY C. M. TANG
	DATE SEPTEMBER, 1969

BLOCKS				
NO.	NO. OF STEPS	DATA REGISTERS	MARK USED	NO. OF STEPS 084
	084	000, 005	0201, 0600, 0601, 0603, 0602	VERIFY NUMBER 636 SET P.C. 000

TO "LEARN" PROGRAM (keyboard to core)	TO RECORD PROGRAM (core to tape)	TO LOAD PROGRAM (tape to core)
1. Set LEARN mode. 2. SET PC to desired step. 3. Index keys in program. 4. Index END PROGRAM as last step in program.	1. Insert tape cartridge. REWIND if necessary. 2. Set TAPE READY. 3. SET PC to first step of program. 4. Index RECORD PROGRAM.	1. Set RUN mode. 2. Insert tape cartridge. REWIND if necessary. 3. Set TAPE READY. 4. SET PC to desired step number. 5. Index LOAD PROGRAM.

PROGRAM DESCRIPTION:

This program can perform simple +, -, x, and ÷ as well as chain operations. This is because the answer of the previous operation is saved, the real part in storage 001 and imaginary part in storage 000. When reading the answer, the real part is in Y register and imaginary part in X register. Same formula is used for entering the complex numbers.

OPERATING PROCEDURE:	EXAMPLE:
1. PRIME ; VERIFY PROGRAM	1. PRIME ; VERIFY PROGRAM
2. Index first number, real part in y and imaginary part in x Set the Toggle Switches to 20 ; Key 01	2. 2 ↑ 5 ; CH SIGN Set the Toggle Switches to 20 ; Key 01
3. Index second number the same way	3. 4 ↑ 3
4. SEARCH + if addition - if subtraction x if multiplication ÷ if division	4. SEARCH +
5. Read answer If (2 - 5i) + (4 + 3i)	5. Read Y | +6.00000000000 X | -2.00000000000

WANG LABORATORIES, INC. 836 NORTH STREET, TEWKSBURY, MASSACHUSETTS 01876
Printed in U.S.A.
700-0243
10-69

PROGRAM TITLE ALGEBRA OF COMPLEX NUMBERS +, -, x, ÷	NUMBER 1004A/MA6
	PAGE 2

If (2 - 5i) - (4 + 3i)

4. SEARCH -
5. Read

Y	-2.0000000000
X	-8.0000000000

If (2 - 5i) x (4 + 3i)

4. SEARCH x
5. Read

Y	+23.0000000000
X	-14.0000000000

If $\frac{(2-5i)}{(4+3i)}$

4. SEARCH ÷
5. Read

Y	-.280000000000
X	-1.04000000000

For chain operations repeat steps 3, 4, and 5

EXAMPLE:

$$\frac{[(1+2i)\ \ (3+4i)] + (6-9i)}{3+4i}$$

2. 1 ↑ 2 ; Key 0201

3. 3 ↑ 4 ; SEARCH x

4 & 5. Read

Y	-5.00000000000
X	+10.0000000000

3. Key 0201

6 ↑ 9 ; CH SIGN

4. SEARCH +

5. Read

Y	+1.00000000000
X	+1.00000000000

3. Key 0201

3 ↑ 4

4. SEARCH ÷

5. Read

Y	+.280000000000
X	-.400000000000-01

700 PROGRAM TITLE: ALGEBRA OF COMPLEX NUMBERS NO. 1004A/MA6 Page 1 of 2

Step	Key	Code	Comment	Step	Key	Code	Comment
00 0	MARK	0408		04 0	÷ DIR	0403	
1	0201	0201		1	REG 03	0003	
2	ST DIR	0404		2	RE DIR	0405	
3	REG 00	0000		3	REG 03	0003	
4	STORE Y	0414		4	MARK	0408	
5	REG 01	0001		5	x	0602	
6	CLEAR X	0715		6	ST DIR	0404	
7	↑	0604		7	REG 003	0003	
8	STOP	0515		8	RE DIR	0405	
9	MARK	0408		9	REG 00	0000	
01 0	-	0601		05 0	ST DIR	0404	
1	CH SIGN	0711		1	REG 04	0004	
2	↓↑	0606		2	↓	0605	
3	CH SIGN	0711		3	X DIR	0402	
4	↓↑	0606		4	REG 00	0000	
5	MARK	0408		5	RE DIR	0405	
6	+	0600		6	REG 01	0001	
7	+ DIR	0400		7	ST DIR	0404	
8	REG 00	0000		8	REG 02	0002	
9	RE DIR	0405		9	↓	0605	
02 0	REG 01	0001		06 0	X DIR	0402	
1	+	0600		1	REG 01	0001	
2	RE DIR	0405		2	RE DIR	0405	
3	REG 00	0000		3	REG 03	0003	
4	STOP	0515		4	↑	0604	
5	MARK	0408		5	X DIR	0402	
6	÷	0603		6	REG 04	0004	
7	CH SIGN	0711		7	RE DIR	0405	
8	ST DIR	0404		8	REG 04	0004	
9	REG 03	0003		9	- DIR	0401	
03 0	x^2	0713		07 0	REG 01	0001	
1	ST DIR	0404		1	RE DIR	0405	
2	REG 05	0005		2	REG 01	0001	
3	↓	0605		3	↓↑	0606	
4	x^2	0713		4	X DIR	0402	
5	+ DIR	0400		5	REG 002	0002	
6	REG 05	0005		6	RE DIR	0405	
7	RE DIR	0405		7	REG 02	0002	
8	REG 05	0005		8	+ DIR	0400	
9	÷	0603		9	REG 00	0000	

700 PROGRAM TITLE: ALGEBRA OF COMPLEX NUMBERS NO. 1004A/MA6 Page 2 of 2

Step	Key	Code	Comment	Step	Key	Code	Comment
08 0	RE DIR	0405		0			
1	REG *00*	*0000*		1			
2	RETURN	0511		2			
3	END PROG	0512		3			
4				4			
5				5			
6				6			
7				7			
8				8			
9				9			
0				0			
1				1			
2				2			
3				3			
4				4			
5				5			
6				6			
7				7			
8				8			
9				9			
0				0			
1				1			
2				2			
3				3			
4				4			
5				5			
6				6			
7				7			
8				8			
9				9			
0				0			
1				1			
2				2			
3				3			
4				4			
5				5			
6				6			
7				7			
8				8			
9				9			

SECTION X WARRANTY, SERVICE AND MAINTENANCE

WARRANTY

Wang electronic equipment is warranted to be free from defects in workmanship and materials for 90 days from delivery to the original purchaser; parts only are warranted for one year, exclusive of labor. Readout tubes, transistors, and fuses are subject to the RETMA guarantee (substituted tubes should be returned to Wang Laboratories). This warranty is in lieu of all other warranties expressed or implied, except as specifically modified in writing by a document signed by an officer of WANG LABORATORIES, INC. Except for such a document, no representative or other person is authorized to represent or assume for WANG LABORATORIES, INC. any warranty liability beyond that set forth herein. Use limits and time between overhaul hours may be specified for mechanical and rotary elements of a Wang system. During the warranty period, Wang equipment is serviced free of charge except for occasional freight cost to and from a service center if equipment is located beyond a 75-mile radius.

POST-WARRANTY SERVICE AVAILABILITY

Wang Service Centers are located in many major cities throughout the world. It is a product service policy to restore the operation of a customer's unit within 24 hours of the service call. For remotely located users, equipment turnaround is normally within one day after arrival at the center. Spare parts, as well as circuit board repair capability are available at all service centers.

ANNUAL MAINTENANCE CONTRACT

An annual maintenance contract is available that consists of adjusting, replacing parts when required and keeping the equipment in first class operating condition. The contract includes all necessary service calls. It does not include repair necessitated by accident, fire, current fluctuations, abuse, or negligence.

POST-WARRANTY SERVICE CALLS WITHOUT MAINTENANCE CONTRACT

All service calls made to customers' facilities not having service contracts will be charged on an hourly basis point to point between the Wang Service Center and equipment location. Automobile charges per mile and material costs will also be included.

NOTE

Users who attempt to repair Wang equipment, without receiving prior Wang equipment training, run the risk of causing further damage to their equipment. Also, and more important, internal equipment voltages are present that could cause severe electrical shock.

IN-HOUSE MAINTENANCE CAPABILITY

Wang Laboratories offers free product familiarization lessons for customers who desire to build up an in-house capability for maintaining their equipment. The customer, of course, is expected to defray the travel and living expenses of his service representative while in training at Wang Laboratories, Tewksbury, Massachusetts.

APPENDIX

TYPING CONVENTIONS

Program 1015A/MA3 VECTOR ANALYSIS is a sample of a 700 library program and is included here to give an example of a program using indirect addressing.

TYPING CONVENTIONS FOR 700 PROGRAM LIBRARY

Certain keyboard instructions have been abbreviated for typing convenience. The following is a listing of the keyboard instructions and their abbreviations.

KEY	ABBREVIATION	CODE
+ DIRECT	+ DIR	0400
– DIRECT	– DIR	0401
X DIRECT	X DIR	0402
÷ DIRECT	÷ DIR	0403
STORE DIRECT	ST DIR	0404
RECALL DIRECT	RE DIR	0405
⇄ DIRECT	EX DIR	0406
SEARCH	SEARCH	0407
MARK	MARK	0408
GROUP 1	GROUP 1	0409
GROUP 2	GROUP 2	0410
WRITE	WRITE	0411
WRITE ALPHA	WRITE A	0412
END ALPHA	END A	0413
STORE Y	STORE Y	0414
RECALL Y	RECALL Y	0415
+ INDIR	+ INDIR	0500
– INDIR	– INDIR	0501
X INDIR	X INDIR	0502
÷ INDIR	÷ INDIR	0503
STORE INDIR	ST INDIR	0504
RECALL INDIR	RE INDIR	0505
⇄ INDIR	EX INDIR 0506	
SKIP IF Y ⩾ X	SKIP IF Y ⩾ X	0507
SKIP IF Y < X	SKIP IF Y < X	0508
SKIP IF Y = X	SKIP IF Y = X	0509
SKIP IF ERROR	SKIP ERROR	0510
RETURN	RETURN	0511

END PROG	END PROG	0512
LOAD PROG	LOAD PROG	0513
GO	GO	0514
STOP	STOP	0515
+	+	0600
−	−	0601
x	x	0602
÷	÷	0603
↑	↑	0604
↓	↓	0605
↑↓	↑↓	0606
$\|x\|$	$\|x\|$	0607
INTEGER X	INT X	0608
π	π	0609
LOG$_{10}$X	LOG$_{10}$X	0610
LOG$_e$X	LOG$_e$X	0611
$\sqrt{x}$	$\sqrt{x}$	0612
10^x	10^x	0613
e^x	e^x	0614
1/x	1/x	0615
0	0	0700
1	1	0701
2	2	0702
3	3	0703
4	4	0704
5	5	0705
6	6	0706
7	7	0707
8	8	0708
9	9	0709
SET EXP	SET EXP	0710
CHANGE SIGN	CH SIGN	0711
	.	0712
x^2	x^2	0713
RECALL RESIDUE	RESIDUE	0714
CLEAR X	CLEAR X	0715

NOTE

For typing convenience the exchange keys, ⇄ **DIRECT** and ⇄ **INDIR**, will be typed as **EX DIR** and **EX INDIR**. **SKIP IF ERROR** has been shortened to **SKIP ERROR**.

In designating the 120 data registers in the KEY column, the register numbers will be preceded by REG.

EXAMPLE

To store into register 58 the coding sheet will appear as follows:

ST DIR
REG 58

A subroutine will be designated in the KEY column by a SR preceding the subroutine code.

EXAMPLE

To address a subroutine beginning with **MARK 0303** the KEY column will appear as follows:

SR 0303

INDEX