## SCALING

The LGP-21 considers all numbers to be within the range $-1 \leq n < +1$. How then, can one use a number like $50.5625_{10}$?

A natural approach would be to move the decimal point two places left, until a number is obtained which can be handled by the computer; namely $.505625_{10}$.

However, this process, known as decimal scaling, is not accurate enough for the LGP-21, and since the computer performs all internal computations in binary form, a preferable approach is to use binary scaling. This means moving the binary instead of the decimal point.

For example, the binary equivalent of $50.5625_{10}$ is

$$110010_{\wedge}1001_2$$

This configuration suggests moving the binary point 6 places to the left to obtain a satisfactory fraction, namely $_{\wedge}1100101001_2$. This process is called q-scaling; in this case, scaling the number at a q of 6 (arithmetically: multiplying by $2^{-6}$).

It would be rather tedious, however, if the programmer had to convert his data — which is normally stated in decimal form — to its binary equivalent. Fortunately, this is not necessary. Appendix C contains a Powers of 2 Table which will tell at a glance that 50.5625 would yield a fraction after a 6-place shift of its binary point.

The table is used in this manner: the left column, labelled "$2^N$", shows that the first power of 2 greater than 50.5625 is 64. Next to the 64, in the center column, is the number 6. This means that $50.5625_{10}$ becomes a fraction if its binary point is shifted left 6 places (or more). Thus, when the number is scaled at a q of 6, it will be within the range of the LGP-21.

It should be emphasized that the appropriate scaling value in the Powers of 2 Table must always be chosen as the next number greater than the one to be converted. For example, if the number above had been 64.000 instead of 50.5625, it could not have been held at a q of 6. The largest number for which this scaling value is valid is 63.999. The rule is not as strict for negative numbers. Both -63.999 and -64 can be held at a q of 6; but -64.001 can not. (It may be recalled that -1 can appear in the LGP-21 unscaled, while +1 can not.)

Sometimes it is desirable to scale numbers which are already fractions, in order to obtain greater arithmetic precision. To do this, a negative scale factor (-q) is specified. For example, the scale factor for the number $.01234_{10}$ is determined by finding the next number larger than .01234 which appears in the right-hand column (labeled "$2^{-N}$") of the Powers of 2 Table-namely .015625. Next to it, in the center column, is a 6. This means that .01234 can be stored at a q of -6 (or any larger q: -5, -4,. . . 0, 1, 2, etc.).

In summary, q-scaling operates as follows:

1. If a number can be expressed exactly in no more than 30 bits and is q-scaled for the LGP-21, it can be stored as an exact number.

2. If a number has to be divided by 2 (that is, multiplied by 1 at a q of 1) to align binary points or avoid overflow, only one of the 30 bits of magnitude of the number is lost.

3. If the binary representation of a number is known, its appearance at any q in a memory location or in the oscilloscope display can be written down. For example, the decimal number 19 has the binary configuration 10011^, since 19 = 16+2+1. In the LGP-21, 19 at a q of 5 would appear as

```
     19
  _________
0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
           ^
```

or at a q of 21 as:

```
                                     19
                                  _________
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0
                                           ^
```

When a number is shifted to the right, the vacated positions on the left side are filled with the original contents of bit position zero, because this is an arithmetic multiply. An example of a positive number shifted right is shown above; for a negative number:

```
            +1.25
+1.25 @ 1   0 ~ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            -1.25
            _______
-1.25 @ 1   1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
               ^
                            -1.25
                            _______
-1.25 @ 9   1 1 1 1 1 1 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                               ^
```

The programmer must know where the binary point is in the result of each arithmetic operation. The rules involved are simple:

1. Addition and Subtraction

   The q's of the operands must be the same, and the q of the result is the same as that of the operands.

   Examples:

```
  6 @ q = 9        15 @ q = 20
 +4 @ q = 9        -9 @ q = 20
 ----------        -----------
 10 @ q = 9         6 @ q = 20
```

   Overflow may occur in which case the computer continues after setting an overflow flag which may be tested by the -Z command.

2. Multiplication

   The q of the product equals the sum of the q's of the two operands.

   Examples:

```
  5 @ q = 5        22 @ q = 26
 x3 @ q = 2        x1 @ q =  3
 ----------        -----------
 15 @ q = 7        22 @ q = 29
```

   The M instruction can be used to shift right. To accomplish this, multiply the number to be shifted by 1 at a q = S, where S is the number of places to shift. In the second example above, 22 is shifted right 3 places because it is multiplied by 1 at a q = 3. The product extends to bit position 30 and is not rounded.

   In M multiplication, no overflow can occur. Bits shifted out of the Accumulator are lost.

   The N instruction can be used to shift left. To N-multiply by 1 @ q = n shifts the number in the Accumulator left 31 - n places without the possibility of overflow, although bits may be shifted out of the Accumulator on the left.

3. Division

The q of the quotient is equal to the q of the dividend minus the q of the divisor. In division it is necessary to determine what q is required for the dividend to insure that the developed q of the quotient will be sufficient to hold the largest expected result. Overflow will occur if the scale of the quotient is not sufficient to cover the answer that is developed.

Examples:

(24 @ 10) ÷ (2 @ 4) = 12 @ 6
(19 @ 17) ÷ (1 @ 2) = 19 @ 15

The D instruction can be used to shift left. To accomplish this, divide the number to be shifted left by 1 at a q = S, where S is the number of binary places to shift. In the second example above, 19 is shifted left 2 places because it is divided by 1 at a q of 2. The quotient is rounded at bit position 30. It is possible to cause overflow when shifting by means of a D instruction.

## DECIMAL CONSTANTS IN A PROGRAM

Since the LGP-21 handles all data and internal computations in binary form, it is desirable to have a program which will read decimally-coded programs, convert them to binary form, and store them in designated locations. Such a program is called a program input routine and is provided to all LGP-21 users.

An LGP-21 program input routine does not accept constants entered in decimal format. They must be entered as instructions or hexadecimal words. For example, 1 at a q of 29 can be conveniently written as the instruction 20001, and 18 at a q of 23 as 21800. Constants which cannot be represented in this way must be written as hexadecimal words (leading zeros may be omitted). For example, 8.75 at a q of 4 must be written as 46000000 on the coding sheet.

Example problem: Calculate $5x^2 + 3x - 7.75 = y$ and store y in 6300 at a q of 10. The value x is less than 10 and is stored in 6301 at a q of 4. The constants 5 @ 3, 1 @ 1, 1 @ 4, 7.75 @ 10, and 3 @ 2 will be in the program.

| PROGRAM INPUT CODES | STOP | LOCATION | INSTRUCTION | | STOP | CONTENTS OF ADDRESS | NOTES |
|---|---|---|---|---|---|---|---|
| | | | OPERATION | ADDRESS | | | |
| | ' | | | | | | |
| | ' | ☒ | | | | | |
| | | 0000 | B | 6301 | ' | Bring x@4 | |
| | | 0001 | M | 6301 | ' | Multiply by x@4; $x^2$ @8 | |
| | | 0002 | M | 0012 | ' | Multiply by 5@3; $5x^2$@ 11 | |
| | | 0003 | D | 0013 | ' ☒ | Shift left 1; $5x^2$@ 10 | |
| | | 0004 | S | 0014 | ' | Subtract 7.75@10; $5x^2$ -7.75 @10 | |
| | | 0005 | H | 6300 | ' | Hold $5x^2$ - 7.75@10 | |
| | | 0006 | B | 6301 | ' | Bring x@4 | |
| | | 0007 | M | 0015 | ' ☒ | Multiply by 3@2; 3x@6 | |
| | | 0008 | M | 0016 | ' | Shift Right 4; 3x@10 | |
| | | 0009 | A | 6300 | ' | Add $5x^2$ -7.75@10; $5x^2$ -7.75 + 3x@10 | |
| | | 0010 | H | 6300 | ' | Store result | |
| | | 0011 | Z | 0000 | ' ☒ | Halt | |
| | | 0012 | 5000 | 0000* | ' | 5@3 in hexadecimal | |
| | | 0013 | 4000 | 0000 | ' | 1@1 in hexadecimal | |
| | | 0014 | 00W8 | 0000 | ' | 7.75 @ 10 in hexadecimal | |
| | | 0015 | 6000 | 0000 | ' ☒ | 3 @ 2 in hexadecimal | |
| | | 0016 | 800 | 0000 | ' | 1@4 in hexadecimal | |

* On the coding sheet, hexadecimal words must be preceded by a Program Input Code (see Figure 3.1) as required by whatever program input routine is being used. Since specific programs are not discussed in this manual, no input codes are shown in the above coding example.

## THE M AND N INSTRUCTION

The N instruction multiplies the contents of the Accumulator by the contents of location m and leaves the least significant half of the result in the Accumulator. The M instruction also causes a multiply operation, but in this case the most significant half of the product is retained.

Multiplying two 30 bit numbers (plus sign) results in a 60 bit product (plus sign), which is in the Extended Accumulator (Figure 5.1). After an M instruction the Accumulator retains the sign and the 30 most significant bits of this product. The remaining bit in the far right of the Accumulator is the spacer bit which is always zero. After using the N instruction for multiplication, the 31st bit of the 60 bit product is in bit position zero of the Accumulator. Bit positions 30 and 31 of the Accumulator are always zero after an N multiply.

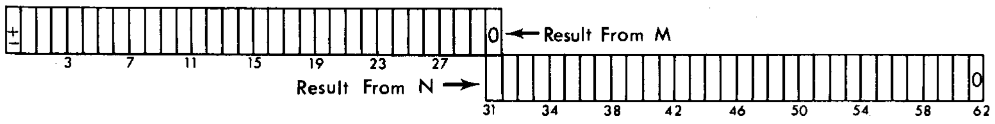

**FIGURE 5.1 Extended Accumulator**

Bit position zero of the Accumulator, after N, has no significance as a sign bit, unless the full 62 bit (plus sign) product contains all O's or all l's up to and including position zero of the result left in the Accumulator.

When an N-multiply instruction is used, the q of the result equals the q of the multiplicand plus the q of the multiplier, minus 31: (x @ 22) x (y @ 14) - 31 = xy @ 5. Examples of multiplication with the N instruction:

(19 @ 20) x (1 @ 28) = 19 @ 17

(120 @ 30) x (10 @ 31) = 1200 @ 30

The first example above demonstrates that N can be used for a left shift of S places by N-multiplying the number to be shifted by 1 at a q of 31 - S. To determine whether to shift left by N-multiplying or by dividing, the following general rule should be used:

If the word to be shifted represents a binarized number and overflow is possible, divide. This allows for overflow detection, with the -Z instruction, which will be explained later.

If the word has a logical meaning and overflow is possible, and if, in effect, a logical shift rather than an arithmetic division is desired, use N.

If no overflow is likely to occur, either D or N can be used to shift left.

The second example above demonstrates that the N instruction can be used to obtain the product at the same q as the multiplicand in the Accumulator, if the multiplier is at a q of 31. However, only even numbers can be held at a q of 31 because position 31-the spacer bit-will be 0.

## VARIATIONS OF THE Z INSTRUCTION

The Z instruction acts as a Halt, Sense Overflow, Sense Branch Switch, or No-operation.

When the address portion of a positive Z instruction is 0000 or 0100, the instruction is interpreted as a Halt. The computer stops in Phase 1 of the instruction following the 20000. The I register will contain the 20000, and the Counter will contain the address of the next instruction. If the track portion of the address is 02 or 03, the Z instruction is treated as a No-operation. That is, the instruction is brought into the I register but is not executed. The Counter is incremented, as usual, and the instruction following the Z is executed normally.

The negative Z instruction is interpreted as a test for overflow. The overflow indicator bit is recorded in the sign position (bit position zero) of the Counter Register. A "1" bit is recorded (i. e. , overflow indicator ON) when overflow occurs. A "0" bit (indicator OFF) signifies that no overflow has occurred. If the overflow bit (not to be confused with position zero of the Z instruction) is ON, the computer turns it OFF and executes the instruction immediately following the -Z; if the overflow bit is OFF, the instruction following the -Z is treated as a No-operation, after which the second instruction following the -Z is executed in the normal manner.

A negative Z instruction is programmed as 800Zn, which results in a 1 bit being placed in position zero of the binary instruction word in memory. This instruction is called the "eight hundred Z" or "minus Z" instruction.

Example of testing for overflow: If the contents of 6308 plus the contents of 6311 results in overflow, go to 0325 for the next instruction. Otherwise, go to 0236.

| PROGRAM INPUT CODES | STOP | LOCATION | INSTRUCTION | | STOP | CONTENTS OF ADDRESS | NOTES |
|---|---|---|---|---|---|---|---|
| | | | OPERATION | ADDRESS | | | |
| | ' | | | | | | |
| | ' | ☒ | | | | | |
| | | 1520 | B | 6308 | ' | | Bring (6308) |
| | | 1521 | A | 6311 | ' | | Add (6311) |
| | | 1522 | 800Z | 0200 | ' | | Test for Overflow |
| | | 1523 | U | 0325 | ' | ☒ | Go to 0325 if Overflow ON. |
| | | 1524 | U | 0236 | ' | | Go to 0236 if Overflow OFF. |
| | | | | | | | |

In the example above, the track address 02 or 03 should be used in the -Z instruction because:

1. If 00 or 01 is used, the computer will halt after turning off overflow. A manual depression of the START switch would then be required to send the computer to Location 1523 for the next instruction. At this time the U0325 would be treated as a No-operation or executed as a branch instruction, depending upon the result of the test.

2. If a track address of 04 or greater is used, the instruction becomes a Sense Overflow and Branch Switch instruction, as explained below.

There are four Branch Switches on the computer console, labeled BS-4, BS-8, BS-16, and BS-32. These switches, which can be manually positioned ON or OFF, operate in conjunction with the track portion of the Z instruction. The computer will test the setting of the Branch Switch indicated by the track address of the Z instruction.

If the Branch Switch is ON, the computer will execute the next instruction in sequence; if the Branch Switch is OFF, the next instruction is treated as a No-operation, and the one following it is executed. For example, if the instruction is 20800 and BS-8 is ON, the instruction following the Z is executed. If BS-8 is OFF, the instruction following the Z is treated as a No-operation, and the one following that is executed normally. Several Branch Switches may be tested with one Z instruction by making the track address equal to the sum of the numbers of the switches. For example, 22800 will cause the computer to test BS-16, BS-8, and BS-4. If all the switches are ON, the next instruction will be executed; if any switch is OFF, the next instruction will be treated as a No-operation.

The Sense Overflow and Sense Branch Switch can be combined (-Z, plus track portion which specifies a Branch Switch). In this case, the next instruction will be executed if the overflow bit and all referenced Branch Switches are ON. If any one of these is OFF, the next instruction will be treated as a No-operation.

This discussion can be summarized as follows:

| Instruction | Interpretation |
|---|---|
| zoooo<br>Z0100 | Halt |
| 20200<br>20300 | No-operation |
| 20400 through 26000 | Sense Branch Switches; skip on no match. |
| -20000<br>-Z0100 | Sense Overflow and halt. |
| - 20200<br>-20300 | Sense Overflow only; skip on no overflow. |
| - 20400 through - 26000 | Sense Overflow and Branch Switches |

## TRANSFER CONTROL SWITCH AND THE T INSTRUCTION

On the computer console is a switch labeled "TC" (Transfer Control), which can be manually positioned to ON or OFF. This switch operates in conjunction with the negative T instruction, programmed 800T, which results in a "1" bit in position zero of the T instruction. This instruction is referred to as the "eight hundred T" or "minus T" instruction. When a -T instruction is executed, a transfer to m will occur to obtain the next instruction if the Transfer Control switch is ON or the Accumulator contains a negative word. If the Transfer Control switch is OFF, the -T instruction will operate normally; that is, a transfer to m will occur if the Accumulator contains a negative word. Therefore, if this instruction is to be used for testing the position of the Transfer Control switch only, the Accumulator must contain a positive word at the time the instruction 'is executed.

## THE E INSTRUCTION

The Extract instruction allows "masking" part of the word in the Accumulator, so that only the desired portion is retained. The masked out portion of the word is set to binary zeros; the word in location m is called the mask. Where the mask contains O's, the corresponding Accumulator bits are set to O's; where the mask contains l's, the corresponding Accumulator bits are left unchanged.

Examples: Assuming that the initial contents of the Accumulator and the contents of 2315 are as shown and that the computer executes an E2315, these would be the results:

First Example

| | |
|---|---|
| Contents of 2315 (the Mask) | 000000000000000011111111111110 |
| Initial contents of Acc. | 001101101010110001010000011101100 |
| Final contents of Acc. | 000000000000000001010000011101100 |

Second Example

| | |
|---|---|
| Contents of 2315 (the Mask) | 111100001111000011110000011110000 |
| Initial contents of Acc. | 011010010001000000001111111110100 |
| Final contents of Acc. | 011000000001000000000000011110000 |

This instruction enables the programmer to "pack" and "unpack" computer words which contain more than one field of data.

Coding example: Location 0523 contains rate of pay in pennies at a q of 15 and hours worked at a q of 30. Compute gross pay (rate x hours) in pennies and store the result in 0563 at a q of 15.

| PROGRAM INPUT CODES | STOP | LOCATION | INSTRUCTION | | STOP | CONTENTS OF ADDRESS | NOTES |
|---|---|---|---|---|---|---|---|
| | | | OPERATION | ADDRESS | | | |
| | | 0000 | B | 0523 | | | Bring word from 0523 |
| | | 0001 | E | 0009 | | | Keep hours worked |
| | | 0002 | H | 6300 | | | Save hours worked in temporary storage |
| | | 0003 | B | 0523 | | | Bring word from 0523 |
| | | 0004 | E | 0010 | | | Keep rate of pay @ 15 |
| | | 0005 | M | 0011 | | | Shift right 1 to q = 16 |
| | | 0006 | N | 6300 | | | (Rate of pay @ 16) x (hours @ 30) = pay @ 46 - 31 = 15 |
| | | 0007 | H | 0563 | | | Store gross pay in 0563 @ q = 15 |
| | | 0008 | Z | 0000 | | | Halt |
| | | 0009 | | WWWQ | | | Hexadecimal representation of mask |
| | | 0010 | WWWW | 0000 | | | Hexadecimal representation of mask |
| | | 0011 | 4000 | 0000 | | | 1 @ q = 1. |

Notice that the instruction sequence B0523, E0009 yields exactly the same result as B0009, E0523. The mask can be in the Accumulator, and the word which is to be edited can be "extracted" from it, if this is more convenient.