IBM

650 MDDPM Additional Features

Indexing Accumulators

Floating-Decimal Arithmetic

Advanced Write-up

Form 22-6258-0

Copyright 1955 by
International Business Machines Corporation
590 Madison Avenue, New York 22, N. Y.
Printed in U. S. A.
Form 22-6258-0

TYPE 650 ADDITIONAL FEATURES

INDEXING ACCUMULATORS

MANY problems require the same operations to be performed repetitively as when processing ordered arrays of data. When this is the case, a large amount of address arithmetic must be done to modify instructions so that they will operate upon the proper data. Index registers are devices that will automatically modify addresses and greatly facilitate the necessary address arithmetic.

Three indexing accumulators are available for the Type 650 MDDPM. Each of these incorporate all the characteristics usually found in index registers, plus the ability to be used as separate accumulators. As accumulators they may be used for accumulating small totals, holding group multipliers, or as small high-speed storage devices. Programming is simplified, the number of instructions required is reduced, and, therefore, programming errors are reduced. Because fewer instructions are executed, the problem solution time will be less. Also, the logic of a program using indexing accumulators is simpler than the logic for a corresponding non-indexed program. This, of course, eases the burden on the programmer and tends toward faster, more accurate programming.

Three indexing accumulators (I.A.) are provided for the Type 650. Each index accumulator retains four decimal digits and an associated algebraic sign. Factors may be added to or subtracted from the contents of an index accumulator, or new factors may be inserted in an index accumulator by reset add or reset subtract operations. It is possible to test each index accumulator for a zero or non-zero, or a positive or negative state by means of branch operations. Each indexing accumulator is addressable so that its contents may be used as a factor in other operations. The primary use of the indexing accumulator, however, is to modify addresses of instructions automatically.

ADDRESSES OF INDEXING ACCUMULATORS

THE ADDRESSES assigned to the indexing accumulator are as follows:

| <i>I. A.</i> | Address |
|--------------|---------|
| A | 8005 |
| В | 8006 |
| С | 8007 |

These addresses may be used as the instruction address of any instruction or as the data address of the following instructions: 00-02, 10-11, 14-19, 30-49, 54, 60-61, 64-69, 90-99. The use of 8005, 6, or 7 as the data address of any other operation will cause a storage-selection error. When one of these addresses is used as an instruction address or as the data address of a branch instruction, the next operation executed will be a NO op whose instruction address is the contents of the I.A. addressed. For example, suppose I.A.A. contains +1234 and the operation 65 0100 8005 is given. Following the reset add as specified by code 65, the operation 00 0000 1234 will be executed. Thus, the contents of the I.A.A. specifies that the next instruction is to be executed from location 1234.

Data address 8005, 6, or 7 of the instructions 00-01 may be used, but nothing will happen because these instructions do not use the data address. Because the shift instructions use only the units digit of the data address, an 8005, 6, or 7 data address in codes 30, 31, 35, 36 causes normal shifting of 5, 6, or 7 places, respectively.

The use of 8005, 6, or 7 as the data address of any of the other permissible instructions will cause the contents of the indexing accumulator to be used as a factor in the operation. When used in this manner, the four digits of the indexing accumulator will be in the four low-order digits of a word that has six zeros in the high-order positions. For example, consider the instruction 65 8006 1234. This instruction will cause the contents of I.A.B. to be reset-added into the four low-order positions of the lower accumulator and zeros to be inserted elsewhere. Because the addition is performed via the distributor, it will also contain six zeros and the contents of the I.A.B. in the four low-order positions. Signs are manipulated just as with any other word.

AUTOMATIC ADDRESS MODIFICATION

THE PRIMARY use of indexing accumulators is to modify addresses automatically by adding the contents of an indexing accumulator to an address. Because the indexing accumulator can contain either positive or negative values, addresses can be modified by adding to them or subtracting from them depending on the sign of the indexing accumulator. Both data addresses and/or instruction addresses can be modified by the contents of any indexing accumulator or by two different indexing accumulators.

It is necessary to tag each address by an indicator so that the Type 650 may know which indexing accumulator should be added to the address. Addresses 2000 through 7999 have been reserved for this purpose. A basic drum address is defined to be one in the range 0000-1999. In order to tag the basic drum addresses, either 2000, 4000, or 6000 is added to indicate that the contents of I.A. A, B or C, respectively, is to be added to the basic drum address. Tagging of high-speed storage addresses is accomplished by adding 200, 400, or 600 to the basic high-speed storage address to indicate the use of I.A. A, B, or C, respectively.

Effective Addresses

The effective address results after a basic address has been modified by the contents of an indexing accumulator. The following table lists all meaningful actual addresses and the resulting effective addresses:

| Actual Address | Effective Address | | | | | |
|----------------|--------------------------------|--|--|--|--|--|
| 0000-1999 | 0000-1999 | | | | | |
| 2000-3999 | 0000-1999 + Contents of I.A. A | | | | | |
| 4000-5999 | 0000-1999 + Contents of I.A. B | | | | | |
| 6000-7999 | 0000-1999 + Contents of I.A. C | | | | | |
| 8000-8003 | 8000-8003 | | | | | |
| 8005-8007 | 8005-8007 | | | | | |
| 8010-8015 | 8010-8015 | | | | | |
| 9000-9059 | 9000-9059 | | | | | |
| 9200-9259 | 9000-9059 + Contents of I.A. A | | | | | |
| 9400-9459 | 9000-9059 + Contents of I.A.B | | | | | |
| 9600-9659 | 9000-9059 + Contents of I.A. C | | | | | |

Meaningful Addresses

The effective address determined as indicated in the foregoing table must be a meaningful address to the operation called for. The following table lists the possible address that may be used with each meaningful operation code:

| A | ddresses | Code 1 | Description . |
|----------|----------------|--------------------|--|
| 0 | 000-1999 | D Dr | um |
| | 000-1999 | | ithmetic unit and |
| _ | | | console switches |
| 8 | 005-8007 | | dexing accumulator |
| | 010-8015 | | pes |
| 9 | 000-9059 | | gh-speed storage |
| ٠. | | Meaningful | |
| Op | e Abbvr. | Data Address | Nama |
| Cou | e About. | | |
| 00 | NO OP | | No Operation |
| 01 | STOP | D,A,I,T,H | Stop |
| 02 | FASN | D,A,I,H | Floating Add Suppress Normalization |
| 03 | RCT | T | Read Check Tape Record |
| 04 | RT | Ŷ | Read Tape Record |
| 05 | RTA | T | Read Alphamerical Tape |
| | | | Record |
| 06 | WT | T | Write Tape Record |
| 07 | WTA | T | Write Alphamerical Tape |
| 08 | LBB | D | Record Load Buffer Block |
| 09 | LB | Ď | Load Buffer |
| 10 | AU | D,A,I,H | Add Upper |
| 11 | SU | D,A,I,H | Subtract Upper |
| 12 | Not Used | | |
| 13 | Not Used | | D: 11 |
| 14 | DIV | D,A,I,H | Divide |
| 15 16 | AL SL | D,A,I,H D,A,I,H | Add Lower Subtract Lower |
| 17 | AABL | D,A,I,H | Add Absolute Lower |
| 18 | SABL | D,A,I,H | Subtract Absolute Lower |
| 19 | MULT | D,A,I,H | Multiply |
| 20 | STL | D,H | Store Lower |
| 21 | STU | D,H | Store Upper |
| 22 | ST DA ST IA | D,H | Store Data Address Store Instruction Address |
| 23 | ST D | D,H D,H | Store Distributor |
| | BNTS | D,A,I,H | Branch No Tape Signal |
| 26 | | | |
| 27 | SET | H | Set Buffer Address |
| 28 | ST BB | D | Store Buffer Block |
| 29 | ST B | DAITH | Store Buffer |
| 30 | SRT | D,A,I,T,H | Shift Right Shift and Round |
| 31 32 | SRD FA | D,A,I,H | Floating Add |
| 33 | FS | D,A,I,H | Floating Subtract |
| 34 | FD | D,A,I,H | Floating Divide |
| 35 | SLT | D,A,I,T,H | Shift Left |
| 36 | SCT | D,A,I,T,H | Shift and Count |
| 37 | FAAB | D,A,I,H | Floating Add Absolute |
| 38 39 | FSAB FM | D,A,I,H D,A,I,H | Floating Subtract Absolute |
| 40 | BNZA | D,A,I,H | Floating Multiply Branch Non-Zero A |
| 41 | BMNA | D,A,I,H | Branch Minus A |
| 42 | BNZB | D,A,I,H | Branch Non-Zero B |
| 43 | BMNB | D,A,I,H | Branch Minus B |
| 44 | BRNZU | | Branch Non-Zero Upper |
| 45 46 | BRNZ BRMIN | D,A,I,H D,A,I,H | Branch Non-Zero Branch Minus |
| 47 | BROV | D,A,I,H | Branch Overflow |
| 48 | BNZC | D,A,I,H | Branch Non-Zero C |

| | | | Meaningful | |
|---|----------------|-----------------|--------------------|--|
| | Οþ | | Data | |
| | Code A | bbvr. | Address | Name |
| | 49 BM | INC | D,A,I,H | Branch Minus C |
| | 50 AA | | D,A,H | Add A |
| | 51 SA | | D,A,H | Subtract A |
| | 52 AB | | D,A,H | Add B |
| | 53 SB | | D,A,H | Subtract B |
| | | | D,A,I,H T | Branch No End of File Rewind |
| | | ľM | T | Write Tape Mark |
| - · · · · · · · · · · · · · · · · · · · | 57 BSI | | τ̈́ | Backspace |
| | 58 AC | _ | D,A,H | Add C |
| | 59 SC | | D,A,H | Subtract C |
| | 60 RA | | D,A,I,H | Reset Add Upper |
| | 61 RS | | D,A,I,H | Reset Subtract Upper |
| | - | ot Used | | |
| | | ot Used VRII | D,A,I,H | Divide Reset Upper |
| | 65 RA | | D,A,I,H | Reset Add Lower |
| | 66 RS | | D,A,I,H | Reset Subtract Lower |
| | 67 R.A | AABL | D,A,I,H | Reset Add Absolute Lower |
| | | ABL | D,A,I,H | Reset Subtract Absolute Lower |
| | 69 LD | | D,A,I,H, | Load Distributor |
| | 70 RI | | D,H | Read to Input Storage 1 Write (Punch or Print) from Output Storage 1 |
| | 71 W | | D,H D,H | Read Conditional to Input Storage 1 |
| | 73 RI | | D,H | Read to Input Storage 2 |
| | 74 W | | D,H | Write (Punch or Print) from Output Storage 2 |
| | 75 RC | | D,H | Read Conditional to Input Storage 2 |
| | 76 RI | | D,H | Read to Input Storage 3 |
| | 77 W. | | D,H | Write (Punch or Print) from Output Storage 3 |
| | 78 RC | | D,H | Read Conditional to Input Storage 3 |
| | 79 No 80 RA | ot Used | D,A,H | Reset Add A |
| | 81 RS | | D,A,H | Reset Subtract A |
| | 82 R.A | | D,A,H | Reset Add B |
| | 83 RS | SB | D,A,H | Reset Subtract B |
| | 84 TI | | D,H | Table Lookup |
| | | ot Used | | |
| | | ot Used | | |
| | | ot Used AC | D,A,H | Reset Add C |
| | 89 RS | | D,A,H | Reset Subtract C |
| | | | D,A,I,H | Branch Distributor Digit 10 |
| | | XD 1 | D,A,I,H | Branch Distributor Digit 1 |
| | | RD 2 | D,A,I,H | Branch Distributor Digit 2 |
| | | ED 3 | D,A,I,H | Branch Distributor Digit 3 |
| | | D 4 | D,A,I,H | Branch Distributor Digit 4 |
| | | ND 5 | D,A,I,H D,A,I,H | Branch Distributor Digit 5 Branch Distributor Digit 6 |
| | | | D,A,I,H | Branch Distributor Digit 7 |
| | | ED 8 | D,A,I,H | Branch Distributor Digit 8 |
| | | ED 9 | D,A,I,H | Branch Distributor Digit 9 |

Examples of Address Modification

| Actual | Con | ntents of I. | A. | Indexed | |
|--------------|--------|--------------|--------|--------------|---|
| Instruction | A | <u>B</u> | С | Instruction | Remarks |
| 65 0123 0124 | 0223+ | 0075 - | 0062+ | 65 0123 0124 | No Indexing |
| 65 2123 0124 | 0223+ | | | 65 0346 0124 | Index D by A |
| 65 0123 6124 | | | 0062+ | 65 0123 0186 | Index I by C |
| 65 4123 4124 | | 0075 - | | 65 0048 0049 | Index D and I by B |
| 65 4123 6124 | | 0075 - | 0062+ | 65 0048 0186 | Index D by B and I by C |
| 65 9215 8002 | 0013+ | | | 65 9028 8002 | Index D by A |
| 65 0123 9627 | | | 0015 - | 65 0123 9012 | Index I by C |
| 65 4015 0124 | | 2345+ | | 65 2360 0124 | 2360 causes storage selection error. |
| 65 9210 0124 | 1983+ | | | 65 0993 0124 | D exceeds 10,000; carry is lost. |
| 65 0123 4124 | | 7878+ | | 65 0123 8002 | I becomes 8002. |
| 65 9615 9218 | 1011 - | | 0015 - | 65 9000 8007 | I becomes 8007. |
| 65 2123 0124 | 1011 - | | | 65 9112 0124 | D becomes "negative." Complement 9112 causes storage selection error. |
| 65 2123 0124 | 1111 - | | | 65 9012 0124 | D becomes "negative." Complement is meaningful, however. |

Address modification is accomplished by adding the contents of an indexing accumulator to a basic address. If the contents of the indexing accumulator is positive and the resulting effective address exceeds 9999, the carry would be lost and only the four low-order digits of the sum would be kept as the effective address. If the contents of the indexing accumulator is negative, the address is modified by subtraction. Because subtraction is accomplished by adding the 10's complement, a carry will always occur when the difference is positive. As above, any such carries are lost. If indexing by subtraction should result in a "negative" address, the complement result will not be recomplemented. This may result in a storage selection error if the complement is not a meaningful address.

TESTING OF INDEXING ACCUMULATORS (IA)

THE FOLLOWING are the operations by means of which the contents of the indexing accumulators may be tested.

40 BNZA Branch Non-Zero I.A. A

If I.A. A contains zeros, the next instruction will be taken from the location specified by the instruction address. If the contents of I.A. A are not zero, the next instruction will be taken from the location specified by the data address.

41 BMNA Branch Minus I.A. A

If the sign of I.A. A is plus, the next instruction will be taken from the location specified by the instruction address. If it is minus, the next instruction will be taken from the location specified by the data address.

42 BNZB Branch Non-Zero I.A. B

If I.A. B contains zeros, the next instruction will be taken from the location specified by the instruction address. If the contents of I.A. B are not zero, the next instruction will be taken from the location specified by the data address.

43 BMNB Branch Minus I.A. B

If the sign of I.A. B is plus, the next instruction will be taken from the locations specified by the instruction address. If it is minus, the next instruction will be taken from the location specified by the data address.

48 BNZC Branch Non-Zero I.A. C

If I.A. C contains zeros, the next instruction will be taken from the location specified by the instruction address. If the contents of I.A. C are not zero, the next instruction will be taken from the location specified by the data address.

49 BMNC Branch Minus I.A. C

If the sign of I.A. C is plus, the next instruction will be taken from the location specified by the instruction address. If it is minus, the next instruction will be taken from the location specified by the data address.

OPERATIONS UPON INDEXING ACCUMULATORS

THE EFFECTIVE address of those instructions that operate upon indexing accumulators must be 0000-1999, 8000-8003, or 9000-9059. This effective address specifies what data are to be used in the operation. If the effective address is in the range 0000-1999, the paddress has a different significance: It is not used as an address, but it is the actual data or amount used by the operation. If it is in the range 8000-8003, or 9000-9059, the data used by the operation will be the four low-order digits and sign of the storage location specified by the address. The meaning of these statements will become clear in the examples that follow. The instruction address has its usual meaning.

50 AA Add to I.A. A

The data specified by the effective data address will be added to the contents of I.A. A.

These examples show that it is possible

- 1. to add a constant (in the range 0000-1999) to A as in the first two examples
 - 2. to add A to itself as in the third example
- 3. to add A to itself and to another constant (in the range 0000-1999) as in the fourth example...
 - 4. to add B to A as in the fifth example
- 5. to add B to A and to another constant as in the sixth example
- 6. to add to A from the four low-order positions of a high-speed storage location as in the next three examples

The last example illustrates how addresses are modified before the operation is executed. Thus, D and I are increased by 0500 before the contents of I.A. A are modified.

As described previously the effective address must be type D, A, or H. The operations performed to obtain this effective address are exactly the same here as with any other kind of instruction. The rules regarding carries and complement (negative) addresses still apply. The final addition to I.A. A is algebraic, when the effective address is type A or H and all normal rules regarding signs apply. If, however, the effective address is Type D, this address is always treated as though it were plus, because the program register carries no sign.

Examples:

| Actual Indexed | | Contents of I.A. A | | Contents | Contents of 8000-8003 or |
|----------------|--------------|-----------------------|--------|-----------|-----------------------------|
| Instruction | Instruction | Before | After | of I.A. B | 9000-9059 |
| 50 0001 0123 | 50 0001 0123 | 0500+ | 0501+ | | |
| 50 1623 0123 | 50 1623 0123 | 0500+ | 2123+ | | |
| 50 2000 0123 | 50 0500 0123 | 0500+ | 1000+ | | |
| 50 2156 0123 | 50 0656 0123 | 0500+ | 1156+ | | |
| 50 4000 0123 | 50 0111 0123 | 0500+ | 0611+ | 0111+ | |
| 50 4265 0123 | 50 0154 0123 | 0500+ | 0654+ | 0111 - | |
| 50 8002 0123 | 50 8002 0123 | 0500+ | 1611+ | | 7777771111+ |
| 50 9007 0123 | 50 9007 0123 | 0500+ | 7277 - | | 1111117777 - |
| 50 9407 0123 | 50 9004 0123 | 0500+ | 1734+ | 0003 - | 0202021234+ |
| 50 2156 2123 | 50 0656 0623 | 0500+ | 1156+ | | |

All other instructions that are described in the following paragraphs operate in an analogous manner. A few examples for each are given. The general rules given are completely sufficient for working out any conceivable conditions.

52 AB Add to I.A. B

The data specified by the data address will be added to the contents of I.A. B.

58 AC Add to I.A. C

The data specified by the data address will be added to the contents of I.A. C.

51 SA Subtract from I.A. A

The data specified by the data address will be subtracted from the contents of I.A. A.

53 SB Subtract from I.A. B

The data specified by the data address will be subtracted from the contents of I.A. B.

59 SC Subtract from I.A. C

The data specified by the data address will be subtracted from the contents of I.A. C.

The following operations are analogous to the previous group except that the index accumulator is reset to zero before the data are added or subtracted into it.

80 RAA Reset Add to I.A. A

I.A. A will be reset to zero, and the data specified by the data address will be added to it.

Examples:

| Actual | Indexed | Contents of I.A. A | | of I.A. A | | Contents | Contents of 8000-8003 or |
|--------------|--------------|--------------------|--------|-----------|--------------|----------|-----------------------------|
| Instruction | Instruction | Before | After | of I.A. B | 9000-9059 | | |
| 51 0001 0123 | 51 0001 0123 | 0500+ | 0499+ | | | | |
| 51 2000 0123 | 51 0500 0123 | 0500+ | 0000+ | | | | |
| 51 4250 0123 | 51 0125 0123 | 0500+ | 0375+ | 0125 - | | | |
| 51 8000 0123 | 51 8000 0123 | 0500+ | 7390 - | | 1234567890+ | | |
| 51 9207 0123 | 51 9057 0123 | 0050+ | 7940+ | | 1234567890 - | | |

Examples:

| Actual | Indexed | Contents of I.A. A | | Contents | Contents of 8000-8003 or |
|--------------|--------------|--------------------|-------|-----------|-----------------------------|
| Instruction | Instruction | Before | After | of I.A. C | 9000-9059 |
| 80 0000 0123 | 80 0000 0123 | 1234 - | 0000+ | | |
| 80 1520 0123 | 80 1520 0123 | 1234- | 1520+ | | |
| 80 6000 0123 | 80 0175 0123 | 1234 - | 0175+ | 0175+ | |
| 80 6525 0123 | 80 0350 0123 | 1234 - | 0350+ | 0175 - | |
| 80 9027 0123 | 80 9027 0123 | 1234 - | 5021- | | 001212345021 - |

82 RAB Reset Add to I.A. B

I.A. B will be reset to zero, and the data specified by the data address will be added to it.

88 RAC Reset Add to I.A. C

I.A. C will be reset to zero, and the data specified by the data address will be added to it.

81 RSA Reset Subtract from I.A. A

I.A. A will be reset to zero, and the data specified by the data address will be subtracted from it.

83 RSB Reset Subtract from I.A. B

I.A. B will be reset to zero, and the data specified by the data address will be subtracted from it.

89 RSC Reset Subtract from I.A. C

I.A. C will be reset to zeros, and the data specified by the data address will be subtracted from it.

Examples:

| Actual | Indexed | Contents of I.A. A | | Contents | Contents of 8000-8003 or |
|--------------|--------------|-----------------------|--------|-----------|-----------------------------|
| Instruction | Instruction | Before | After | of I.A. C | 9000-9059 |
| 81 1234 0123 | 81 1234 0123 | 0527+ | 1234 - | | |
| 81 7015 0123 | 81 1927 0123 | 0527+ | 1927 - | 0912+ | |
| 81 7015 0123 | 81 0103 0123 | 0527+ | 0103 - | 0912 - | |
| 81 9059 0123 | 81 9059 0123 | 0527+ | 2301+ | | 0123012301 - |

FLOATING-DECIMAL ARITHMETIC

THE PRESENT operations of the Type 650 have proved to be satisfactory for the great majority of problems that it has solved. There are, however, many problems involving lengthy, complex calculations that require extensive analysis to determine the size and range of intermediate and final quantities. This analysis and the subsequent scaling of these quantities frequently require a larger percentage of the total time required to solve the problem than the actual calculations.

Floating-decimal arithmetic circumvents this difficulty by tagging each number with a 2-digit characteristic. This characteristic specifies where the decimal point should be. Use of this technique virtually eliminates the need for scaling numbers.

Floating-decimal numbers in the Type 650 look exactly like fixed-point numbers. The only difference between them is the way in which the arithmetic unit interprets them when a floating-decimal operation is needed. Seven new instructions have been added to operate upon floating-decimal numbers. They are add, subtract, add absolute, subtract absolute, multiply, divide, and non-normalize add. Whenever one of these operations is called for, the numbers operated upon are interpreted as follows:

$$(M,C) = \underbrace{\underbrace{.xxxxxxx}_{M} \underbrace{xx}_{C} \underbrace{\pm}_{Sign}}_{\text{of } M}$$

The mantissa M is eight decimal digits in length. The decimal point of the mantissa lies to the left of the eighth digit. The sign of the number is always associated with the mantissa. Thus, the range of the mantissa is

$$0.1 \leq |M| < 1.0$$

The exponent e is a two-digit integer in the range

$$-50 \le e \le 49$$

Because the sign is associated with the mantissa, it cannot be used to indicate the exponent sign. By

adding 50 to the exponent, a positive number C in the range

$$0 \le C = e + 50 \le 99$$

is obtained. It is the two-digit characteristic C that is carried as a tag to specify where the decimal point of the number really is.

To summarize then, the fixed point number, N, being represented by the floating point number (M,C,) is determined by

$$N = M \times 10^{C-50}.$$

For example:

1.0 would be represented as 1000000051.

Because there is no difference between fixed and floating-decimal numbers, fixed-point operations can be performed upon floating-decimal numbers if desired. For example, it is possible to test the floating-point number to determine whether it is zero or non-zero; positive or negative. It is up to the programmer to determine what he wishes to do and to write the proper sequence of instructions to perform that operation. Similarly, the characteristic can be separated from the mantissa by shifting and examining. It can be modified by programming; whatever else is desired can be done. This facility for operating upon numbers with either type of arithmetic provides great flexibility.

OPERATION CODES

THE FOLLOWING descriptions tell how each of the seven new instructions function. Any operation that results in a zero mantissa will force a zero exponent.

In all of these floating-decimal arithmetic operation codes, the prior amount in the lower accumulator is ignored by the operation. Also the lower accumulator will be set to zeros upon completion of the operation and its sign will be the sign of the result in the upper accumulator. Either a divide overflow or an exponent overflow stops the machine.

32 FA Floating Add

The floating-decimal number specified by the data address is added to the floating-decimal number in the upper accumulator. The rounded result will be retained in the upper accumulator.

33 FS Floating Subtract

The floating-decimal number specified by the data address is subtracted from the floating-decimal number in the upper accumulator. The rounded result will be retained in the upper accumulator.

34 FD Floating Divide

The floating-decimal number in the upper accumulator is divided by the floating-decimal number specified by the data address. The rounded quotient will be retained in the upper accumulator.

37 FAAB Floating Add Absolute

The absolute value of the floating-decimal number specified by the data address is added to the floating decimal number in the upper accumulator. The rounded result will be retained in the upper accumulator.

38 FSAB Floating Subtract Absolute

The absolute value of the floating-decimal number specified by the data address is subtracted from the floating-decimal number in the upper accumulator. The rounded result will be retained in the upper accumulator.

39 FM Floating Multiply

The floating-decimal number in the upper accumulator is multiplied by the floating-decimal number specified by the data address. The rounded result is retained in the upper accumulator.

02 FASN Floating Add, Suppress Normalization

This code operates exactly the same as 32 (FA) except that the normalization, which occurs after adding the shifted numbers, is suppressed. This makes it possible to attach the same exponent to a group of numbers for fixed-point output.

OPERATING SPEEDS

THE TIMES required to execute the foregoing operations are essentially the same as for corresponding fixed-point operations. Because multiply uses only an 8-digit multiplier, it will be faster than fixed-point multiply. The add type operations will vary in length depending upon the number of shifts required to line up the decimal points or to normalize the sum. The minimum time is approximately 1.0 milliseconds and the maximum 2.4 milliseconds; a good average is probably 1.7 milliseconds. Thus, floating-point add is about half as fast as fixed-point add when optimum programmed. Random programming, of course, requires the same time for both types of addition, namely 5.2 milliseconds.



Form 22-6258-0 (11-56:10M)