

IBM**Data Processing Techniques****Analysis** **System/360 Pointers****Programming Techniques**

This manual of System/360 pointers is devoted to examples illustrating various coding techniques. It not only shows the use of some of the features of the assembly language but more importantly points out the power of the System/360 organization in the solution of common problems or parts thereof. Although many of the examples are slanted toward the mathematician, there are those of sufficiently general interest to provide knowledge for the commercially oriented.

Programming **Scientific** 

FOREWORD

The examples presented in this manual are small in size and are intended for a reader who is interested in an introduction to System/360 programming. While studying the various pieces of code, it is advisable to have access to the following manuals:

System/360 Principles of Operation (A22-6821-1) -- referred to as "OM"

System/360 Operating System -- Assembly Language (C28-6514-1) -- referred to as "AM"

References are made to these manuals in the subsequent pages.

There are three basic sections in this document. In the first two sections, the basic instruction set is illustrated through simple examples. It is intended that the examples increase in complexity, each tending to illustrate some particular point relative to a coding technique. Pertinent comments are included. The final section shows complete problems, including the necessary assembly language parameters to produce a running program. The problems of this section have actually been run on a machine.

The programs shown represent "good" coding practices. They have not been optimized to their fullest. This optimization is left as an exercise to the reader. Where optional methods of approach can be taken, they are shown and discussed.

CONTENTS

1.	<u>Fixed-Point, Logical and Branch Instructions</u>	1
1.1	Use of Multiple Registers for Storing Intermediate Results	1
1.2	Evaluation of a Simple Arithmetic Expression	1
1.3	Cyclic Purmutation of a Word Group Using Multiple Register Commands .	2
1.4	Clearing of a Space in Storage Using Loop Control Techniques	3
1.5	Extraction of one Element of a Matrix — Two Dimensional Table Lookup .	5
1.6	Cyclic Bit Shifting	5
1.7	Sum of Squares Using Double-Precision Fixed-Point Arithmetic	6
1.8	Transposition of a Square Matrix Using Address Constants	7
1.9	Length of an Unknown File Using Byte-Handling Capabilities and Multiple Registers	8
1.10	Table Lookup Using the Translate Instruction	10
2.	<u>Floating-Point Instructions</u>	12
2.1	Polynomial Evaluation	12
2.2	Separation of Integer and Fraction Parts of Floating-Point Numbers Using Unnormalized Instructions.	12
2.3	Extracting the Integer Part of a Floating-Point Number	13
2.4	Converting a Fixed-Point Number to a Floating-Point	15
2.5	Double-Precision Square Root (SQRTD).	16
2.6	Matrix Addition	17
2.7	Product of Two Square Matrices.	19
3.	<u>Complete Problems.</u>	20
3.1	Sorting on the Basis of Subfields.	20
3.2	Removal of Keywords.	21
3.3	Prime Number Generator	23
3.4	Min-Max Problem	24
3.5	Double-Precision Binary-to-Decimal Conversion	25
3.6	Scanning a Message for Syntactical Errors	26

1. FIXED-POINT, LOGICAL AND BRANCH INSTRUCTIONS

1.1 USE OF MULTIPLE REGISTERS FOR STORING INTERMEDIATE RESULTS

The fixed-point numbers a, b, c, d are contained in the general purpose registers GPR1, GPR2, GPR3, and GPR4, respectively. Calculate the quantities

$$r = a+b + |c-d|$$

and

$$s = a+b - |c-d|$$

putting r and s in GPR5 and GPR6, respectively.

AR	1, 2	$a+b$ Register-to-register add.
SR	3, 4	$c-d$ Register-to-register subtract.
LPR	5, 3	$ c-d $ Make the sign positive.
AR	5, 1	$r=a+b + c-d $
LNR	6, 3	$- c-d $ Make sign negative.
AR	6, 1	$s=a+b - c-d $

Comments

- Fixed-point arithmetic is executed in the general purpose registers. Numbers in these registers are treated as signed integers (O.M., p. 23).
- The above example shows the advantage of multiple accumulators. Intermediate results can be contained in registers, and no time-consuming reference to storage has to be made. (We assume that the numbers are such that no overflow occurs during any of the operations.)

1.2 EVALUATION OF A SIMPLE ARITHMETIC EXPRESSION

Evaluate $r = -2.0 * (a+b) * d/c$, where a, b, c, d are the contents of general purpose registers GPR1, GPR2, GPR3, and GPR4, respectively. Place the result in GPR5.

LCR	7, 1	-a. Complement a and load into GPR7.
SR	7, 2	$-(a+b)$. Subtract b from $-a$.
MR	6, 4	$-(a+b)*d$. Note multiply and divide conventions in MR and DR.
DR	6, 3	$-(a+b)*d/c$

LR	5,7	Quotient to GPR5.
SLA	5,1	Shift to multiply by 2.

Comments

- a. This problem illustrates the positioning of operands and result in the fixed-point multiply and divide operations (O.M., pp. 29-30).
- b. Multiplication or division by a power of 2 for a number in fixed-point format can be accomplished by shifting. This is generally more efficient than a multiplication. Note, in the use of shifting for division, that the 2's complement notation implies a different rounding, as the digits shifted out are always positive. Thus a right shift on +25 yields +12, but a right shift on -25 yields -13. A division of -25 by 2 yields -12, with -1 as remainder.
- c. Fixed-point arithmetic treats operands in the registers as if they were integers. The multiply instruction takes the multiplicand from an odd register and develops a double-length result in an even/odd register pair. For division the dividend is taken to be a double-length number situated in an even/odd register pair. The quotient is obtained in the odd register, while the remainder is in the even register (O.M., pp. 29-30).

1.3 CYCLIC PERMUTATION OF A WORD GROUP USING MULTIPLE REGISTER COMMANDS

Given the quantities $A_1, A_2 \dots A_{16}$ in fullword locations starting at LOC. Cyclically permute the information such that it is stored in the sequence $A_4, A_5 \dots A_{16} A_1 A_2 A_3$.

Method 1

LM	0, 15, LOC	Load all registers with A_1 through A_{16} .
STM	0, 2, LOC+52	Store A_1, A_2, A_3 at end of LOC area.
STM	3, 15, LOC	Store A_4 through A_{16} .

Comments

LOC+52 refers to the byte whose address is LOC+52. Since there are four bytes per word, this refers to the 14th word of the group.

Method 2

LM	13, 15, LOC	Load registers with A_1, A_2, A_3 .
LM	0, 12, LOC+12	Load register with A_4 through A_{16} .
STM	0, 15, LOC	Store multiple in permuted order.

Method 3

LM	13, 12, LOC	Load registers with A ₁ through A ₁₆ .
STM	0, 15, LOC	Store multiple in permuted order.

Comments

- a. Method 3 makes use of the wraparound feature of the Load Multiple instruction (O.M., p. 26).
- b. The above methods assume that all 16 general purpose registers can be used -- that is, that no base registers are needed to generate address LOC. This confines the numerical equivalent of LOC to less than 4096. Usually, of course, some registers will have to be set aside as base registers, in which case the problem will have to be done in several steps.

1.4 CLEARING OF A SPACE IN STORAGE USING LOOP CONTROL TECHNIQUES

Replace the contents of 25 fullwords starting at LOC with zeros.

Method 1:

	SR	0, 0	Clear GPR0 (see comment a).
	SR	1, 1	Clear GPR1.
	L	2, COUNT	Place a 25 in GPR2.
LOOP	ST	0, LOC(1)	Clear word at LOC indexed by GPR1.
	A	1, FOUR	Increment index by 4.
	BCT	2, LOOP	Reduce count by 1 and test for completion.
	SVC	0	Supervisor call (see comment c).
COUNT	DC	F'25'	See comment b.
FOUR	DC	F'4'	See comment b.

Comments

- a. The fastest and most convenient way of clearing a register is to subtract it from itself.
- b. DC is a pseudo operation for setting up constants in the program. Hence, the steps COUNT and FOUR are not treated as instructions. The F denotes that the constant following it is expressed in 32-bit fixed-point (A.M., p. 37).
- c. SVC 0 is a supervisor call denoting the end of the program. This is the standard way of terminating the execution of a program.

Method 2

Instead of using a BCT to control the loop, one can use a BXLE (O.M., p. 65).

	SR	0,0	Clear GPR0.
	SR	1,1	Clear GPR1.
	L	2,FOUR	Set GPR2 to 4.
	L	3,END	Set end-of-loop test to 96.
LOOP	ST	0,LOC(1)	Clear word at LOC indexed by GPR1.
	BXLE	1,2,LOOP	Add GPR2 to GPR1 and test whether greater than GPR3.
	SVC	0	
FOUR	DC	F'4'	
END	DC	F'96'	End-of-loop test = 25x4-4.

Comments

- This method produces a more efficient program, but uses one more general purpose register.
- The steps L 2, FOUR and L 3, END can be replaced by LM 2, 3, FOUR since the two words to be loaded are adjacent in memory.

Method 3

	LM	0,9,C	Load multiple with zero.
	STM	0,9,LOC	Store zeros in the first ten words.
	STM	0,9,LOC+40	Store zeros in next ten words.
	STM	0,4 LOC +80	Store zeros in remaining five words.
	SVC	0	Return to supervisor.
C	DS	10F'0'	Ten words of zero.

Comments

This method is faster than the first two, but less general.

Method 4

SR	0, 0	Set register 0 to zero.
STC	0, LOC	Store zero into first character of field.
MVC	LOC(99), LOC+1	Propagate zero through the 25 words (100 bytes).

Comments

Method 4 is included here because of its intrinsic interest.

1.5 EXTRACTION OF ONE ELEMENT OF A MATRIX -- TWO-DIMENSIONAL TABLE LOOKUP

Given a matrix of size $M \times N$ (M rows and N columns, $M, N \leq 1000$) stored row-wise in consecutive fullwords beginning with A_{11} in MTRIX. Given also two binary integers p and q in GPR1 and GPR2, respectively. Put the element A_{pq} in GPR3.

L	5, N	Load number of columns into GPR5.	
S	1, ONE	$p-1$	
MR	0, 5	GPR1 contains $(p-1)*N$.	
AR	1, 2	$q+(p-1)*N$	
SLA	1, 2	$4*(q+(p-1)*N)$ Multiply by 4, using a left shift of 2.	
L	3, MTRIX-4(1)	Load element into GPR3 (see comment a).	
SVC	0		
N	DC	F'5'	$N=5$ in this example.
ONE	DC	F'1'	

Comments

- a. The element A_{pq} has byte address $MTRIX + 4*((q-1)+(p-1)*N)$ or $(MTRIX-4)+4*(q+(p-1)*N)$

1.6 CYCLIC BIT SHIFTING

Cyclic left shift seven bit positions the information in the 32-bit fullword LOC and store the result in RES.

SR	2, 2	
L	3, LOC	
SLDL	2, 7	Separate 32 bits of information into 7 and 25 bits.
OR	3, 2	
ST	3, RES	
SVC	0	

Comments

The OR instruction is used to add the seven bits back into register 3. One could also use the ALR instruction to do this, but not an AR instruction. AR would give an incorrect result when the leading bit (sign position) of the shifted number is a 1.

1.7 SUM OF SQUARES USING DOUBLE-PRECISION FIXED-POINT ARITHMETIC

Eight 32-bit integers starting at fullword boundary NUMB are given. Compute the sum of the squares of these integers and store it at SUM in the format sign plus a 63-bit integer. It may be assumed that the sum can be expressed as a 63-bit integer.

	LM	0, 5, REGIS	Set up registers.
LOOP	L	13, NUMB(4)	Load integer.
	MR	12, 13	Square the number.
	ALR	3, 13	Low order of 64-bit sum.
	BC	3, CARRY	Branch if carry out of low-order word.
CONT	AR	2, 12	High order of 64-bit sum.
	BXLE	4, 0, LOOP	Close loop.
	STM	2, 3, SUM	Store result in SUM.
	SVC	0	
CARRY	AR	12, 5	Carry into high-order word by adding 1.
	B	CONT	
REGIS	DC	F'4, 28, 0, 0, 0, 1'	

Comments

- a. Loading of consecutive registers can be done efficiently with a LM instruction.
- b. When a carry out of sign position in ALR instruction occurs, the condition code will be either 2 or 3 (O.M., p. 27). Hence BC3 results in a successful branch whenever there is a carry.

1.8 TRANSPOSITION OF A SQUARE MATRIX USING ADDRESS CONSTANTS

An NxM matrix has fullword elements and is stored row-wise beginning at MATRIX. Create the transpose of this matrix and store it in the same area.

	LM	5, 11, PARAM	
GO	LR	12, 9	column pointer.
	LR	13, 10	Row pointer.
TRANS	L	3, 0(12)	Load a_{ij} .
	L	4, 0(13)	Load a_{ji} .
	ST	3, 0(13)	Store a_{ij} in a_{ji} .
	ST	4, 0(12)	Store a_{ji} in a_{ij} .
	AR	12, 5	Step to next column.
	BXLE	13, 6, TRANS	Step to next row and close loop.
	AR	9, 11	Set up for next column.
	AR	10, 11	Set up for next row.
	BCT	8, GO	
	SVC	0	
PARAM	DC	F'4'	
	DC	A(4*N, MATRIX+4*(N*N-1))	
	DC	A(N-1, MATRIX+4, MATRIX+4*N, 4*(N+1))	

Comments

- a. We want to replace the element a_{ij} of the matrix by a_{ji} and vice versa.
- b. DC type A is used for setting up address constants (A.M., p. 48).

1.9 LENGTH OF AN UNKNOWN FILE USING BYTE-HANDLING CAPABILITIES AND MULTIPLE REGISTERS

Information of unknown length is written in consecutive eight-bit bytes beginning at INFO. Its end is signified by a special character of eight binary 1's. Find the file length (including the special character) in bytes, and put the answer in GPR1.

Method 1

	SR	2,2	Set register to 0.
	L	6, ONE	
	LR	1,6	
	L	3, SPCH	GPR3 has special character.
IC	IC	2, INFO-1(1)	Load one byte of information.
	CLR	2,3	Compare it with GPR3.
	BE	EQUAL	Branch if equal (see comment b).
	AR	1,6	
	BC	15, IC	Unconditional branch (see comment b).
EQUAL	SVC	0	
SPCH	DC	X'000000FF'	
ONE	DC	F'1'	

Comments

- a. The IC instruction can be used to handle one byte at a time.
- b. BC with mask = 15 results in an unconditional transfer. This could also be written in extended mnemonic form (A.M., p. 33) simply as B. Conversely, the BE, which is in extended mnemonic form, could be written as BC8.
- c. DC type X is used for setting up hexadecimal constants (A.M., p. 43).

Method 2

	LM	1, 5, PARAM	
IC	IC	5, 0(1, 4)	
	AR	1, 3	Add to counter.
	CLR	3, 5	Compare with eight binary 1's.
	BC	7, IC	Branch if not equal.
	SVC	0	
PARAM	DC	F'0'	
	DC	XL4'FF'	
	DC	F'1'	
	DC	A(INFO)	
	DC	F'0'	

Comments

- a. The programmer may, if he so desires, specify which base register is to be used. This is shown in the second step. Alternatively, one could write the second step as IC 7, INFO(3) and the assembler would assign the base register.

Method 3

	LM	1, 6, PARAM	LOAD GP registers.
IC	IC	3, 0(1, 6)	Insert character into 3.
	AR	1, 4	Increment index
	BXH	5, 2, IC	See comments
	SVC	0	
PARAM	DC	F'0'	Reg 1 - Index.
	DC	F'0'	Reg 2 - Zero.
	DC	F'0'	Reg 3 - Character from field.

DC	F'1'	Reg 4 - Index increment.
DC	XL4'FF'	Reg 5 - Eight 1's.
DC	A(INFO)	Reg 6 - Base address of field.

Comments

The eight 1's are added to zero, and then compared with the target character in register 3. Since eight 1's will always be higher than any other character, the branch will be taken unless there are eight 1's in the target character. Then the exit will be taken. This method is included to show a variation in use of index control instructions.

Comment On A Possible Method 4

If memory space is not critical, or if speed of function is critical, the most efficient and probably best method is to use the "Translate and Test" instruction. The program is left as an exercise for the reader.

1.10 TABLE LOOKUP USING THE TRANSLATE INSTRUCTION

It is desired to produce a list of eight-bit BCD code from a list of four-bit hexadecimal characters. (This may be used, for example, for printing.) The given list is located at fullword boundary HEX extending through eight doublewords for a total of 128 hexadecimal characters. The output BCD code is to be located at fullword boundary BCD.

	LM	1, 5, REGIS	
NEXT	LH	8, HEX(1)	Fetch four hex characters.
	LR	6, 4	Load GPR6 with 4.
LOOP	SRDL	8, 4	Shift to GPR9 one hex digit.
	SRL	9, 4	Expand to byte length.
	BCT	6, LOOP	
	ST	9, BCD(5)	Store four expanded characters.
	AR	5, 4	
	BXLE	1, 2, NEXT	
	TR	BCD(128), TABLE	Translate entire line.
	SVC	0	

REGIS	DC	A(0, 2, 63, 4, 0)
TABLE	DC	X'F0F1F2F3F4F5F6F7'
	DC	X'F8F9C1C2C3C4C5C6'

Comments

The translate instruction (TR) works like a table-lookup scheme. It translates 256 bytes with just one setup. Here the eight-bit arguments are located at BCD and the eight-bit function bytes are located at TABLE. The translation of each argument byte is completed with the function byte replacing the argument at BCD.

2. FLOATING-POINT INSTRUCTIONS

2.1 POLYNOMIAL EVALUATION

Evaluate the polynomial

$$P(x) = \sum_{K=0}^{20} A_k x^k$$

where x is located at X, a₂₀ at A, a₁₉ at A+4, etc. Store the result (single precision) in POLY.

	SR	1, 1	Clear GPR1.
	LM	2, 3, INCRE	Load index value and limit.
	LE	0, X	Load argument in FPR0.
	SER	2, 2	Zero FP register 2.
LOOP	AE	2, A(1)	Add coefficient of kth term.
	MER	2, 0	Multiply by x.
	BXLE	1, 2, LOOP	
	AE	2, A(1)	Add coefficient of 0th order.
	STE	2, POLY	Store result.
	SVC	0	Return to supervisor.
INCRE	DC	F'4'	
	DC	F'76'	

Comments

The above technique is known as "nesting". $P(x) = (\dots (A_{20}x + A_{19}) x + \dots + A_1) x + A_0$. technique is more efficient than a term by term evaluation.

2.2 SEPARATION OF INTEGER AND FRACTION PARTS OF FLOATING-POINT NUMBERS USING UNNORMALIZED INSTRUCTIONS

The single-precision (32-bit) floating-point number N in location DOG has a small (≤ 6) exponent magnitude. Create two floating-point numbers I, F in CAT and CAT+4 such that

I = an integer

$|F| < 1.0$, sign of F = sign of N

and

I + F = N.

SDR	6, 6	Zero FPR 6 (double).
LDR	2, 6	Zero FPR 2 (double).
LE	6, DOG	Load N.
AW	6, X6	Integer in 1st and fraction 2nd half of FPR6.
LER	2, 6	Load integer into FPR2.
SDR	6, 2	Fraction in FPR6.
AU	6, X0	To force an unnormalized fraction.
STE	2, CAT	Store integer part.
STE	6, CAT+4	Store fractional part.

	SVC	0	
	DC	0D	Place X6 on doubleword boundary.
X6	DC	X'4600000000000000'	Exp 6, fraction 0.
	DC	X'40000000'	Exp 0, fraction 0.

Comments

- a. I and F are generally unnormalized, the exponent of I being 6 and that of F zero.
- b. Unnormalized addition of a number with a zero fraction can be used to force the exponent of number to a predetermined value.

2.3 EXTRACTING THE INTEGER PART OF A FLOATING-POINT NUMBER

The 32-bit single-length floating-point number N in location DOG has a small exponent (≤ 6) and is therefore less than 2^{24} in magnitude. Put the integer part of the number in GPR1 in fixed-point integer form.

Method 1

	LE	4, DOG	
	AU	4, X6	Integer part with exponent of 6 in FPR4.
	STE	4, TEMP	
	L	1, TEMP	Integer part with exponent in GPR1.
	N	1, XZERO	Blank out exp., leaving sign.
	LTR	1, 1	Test whether number is negative.
	BC	10, NOTNEG	Branch if positive or zero.
	N	1, NSIGN	Remove sign.
	LNR	1, 1	Take 2's complement.
NOTNEG	SVC	0	
TEMP	DS	1E	
XZERO	DC	X'80FFFFFF'	Mask to eliminate exponent.
X6	DC	X'46000000'	Floating zero with 6 exponent.
NSIGN	DC	X'7FFFFFFF'	Mask to eliminate sign.

Comments

A negative number in fixed point is represented in two's complement form, while in floating-point the fraction is represented in true form. Hence special steps have to be taken to create the complement if necessary.

Method 2

This method does not use the fixed-point registers.

	LE	4, DOG	Load N into FPR 4.
	AU	4, X6	Integer part with exponent of 6 in FPR4.
	BC	10, NOTNEG	Branch if positive or zero.
	AU	4, ONES	Form 1's complement.
	AU	4, ONE	Form 2's complement.
	STE	4, I	Store result in I.
	MVI	I, X'FF'	Place FF in exponent.
	SVC	0	
NOTNEG	STE	4, I	Store result in I.

	MVI	1, X'00'	
	SVC	0	Blank out exponent.
I	DS	F	Align on fullword boundary
X6	DC	X'46000000'	Unnormalized zero.
ONES	DC	X'46FFFFFF'	Maximum positive floating-point integer.
ONE	DC	X'46000001'	Unnormalized 1.

Method 3

	LE	4, DOG	
	AW	4, MASK	Integer part to right end of FPR4.
	STD	4, TEMP	Does not change condition code.
	L	1, TEMP+4	Does not change condition code.
	BC	10, EXIT	
	LCR	1, 1	
EXIT	SVC	0	
TEMP	DS	1D	
MASK	DC	X'4E00000000000000'	

Comments

- The AW instruction forces the integer part of the number to the extreme right end of the 64-bit floating-point register.
- The condition code set by the AW instruction is not affected by the two subsequent instructions.

Method 4

	LE	4, DOG	
	AU	4, X6	Integer part with exponent of 6 in FPR4.
	STE	4, TEMP	
	L	0, TEMP	Integer part with exponent in GPR0.
	LPR	1, 0	Load absolute value if positive and complement if negative into GPR1.
	SLDA	0, 39	Correct sign still in GPR0.
			Shift quantity against sign and shift out the exponent.
	SRA	0, 7	Shift to proper position.
	SVC	0	
TEMP	DS	1E	
X6	DC	X'46000000'	

Comments

LPR creates the correct quantity except for the exponent part, now extraneous, and sign. The left double shift removes the exponent and regains the sign. The right shift provides the proper offset. Negative numbers will have leading 1-bits as prescribed by the two's complement notation.

2.4 CONVERTING A FIXED-POINT NUMBER TO FLOATING-POINT

GPR0 contains a small integer which is less than 2^{24} in magnitude. The integer is to be converted into a floating-point number and left in FPR0 in double-precision form.

Method 1

	LR	4, 0	
	N	4, SMASK	Separate sign bit and leave in GPR4.
	LPR	3, 0	Load absolute value into GPR3.
	OR	3, 4	Put back sign bit.
	O	3, X6	Tag on exponent of 6.
	ST	3, TEMP	Store into left half of TEMP.
	LD	0, TEMP	Load doubleword with right half of FPR0 zero.
	AE	0, X6	Normalize the number.
	SVC	0	
TEMP	DS	1D	
SMASK	DC	X'80000000'	
X6	DC	X'46000000'	

Comments

The right half of FPR0 may have contained extraneous bits. The use of TEMP as a doubleword avoids an extra zeroing instruction.

Method 2

	LPR	1, 0	Load absolute value into GPR1.
	O	1, MASK	Insert an exponent of 6.
	SLDA	0, 32	Shift fraction and exponent into GPR0. Original sign is preserved.
	ST	0, TEMP	
	LD	0, TEMP	
	AE	0, MASK	To force normalization.
	SVC	0	
TEMP	DC	D'0.0'	
MASK	DC	X46000000'	

Comments

- The LPR instruction creates the absolute value (with eight lead zero-bits) in GPR1, the SLDA instruction shifts the GPR(0, 1) pair to result in the correct sign-magnitude notation. The additional instructions move the result to FPR0 and perform normalization.
- AE can be used to force normalization since the right-hand side of FPR0 is known to be zero.

2.5 DOUBLE-PRECISION SQUARE ROOT SUBROUTINE

Given a double-precision number x in FPRO, create the double-precision square root in FPRO

Algorithm: let $x=16^E.f$ where $\frac{1}{16} \leq f < 1$

then an approximation y_0 to $y = \sqrt{x}$ can be obtained by the linear expression

$$y_0 = \left(\frac{8f}{9} + \frac{2}{9} \right) 16^{\lfloor E/2 \rfloor} \text{ when } I \text{ is even}$$

$$y_0 = \left(\frac{32f}{9} + \frac{8}{9} \right) 16^{\lfloor E/2 \rfloor} \text{ when } I \text{ is odd}$$

(Here $\lfloor E/2 \rfloor$ denotes the integer part of $E/2$.)

It can be shown that such an approximation has a relative error which is nowhere greater than $1/9$. Therefore, four Newton-Raphson iterations of the form

$$Y_{n+1} = \frac{1}{2} \left(Y_n + \frac{X}{Y_n} \right)$$

are sufficient to give better than 56-bit accuracy.

Conventions

In order to gain speed, the loop for the Newton-Raphson iterations was expanded and single precision used whenever possible. In order to achieve the multiplication by $1/2$, the HALVE instruction is used when possible. However, this cannot be done on the last step, since HALVE produces an unnormalized result. This can be overcome by using a multiply by $1/2$ on the last step. Alternatively one could use

$$Y_4 = Y_3 + \frac{1}{2} \left(\frac{N}{Y_3} - Y_3 \right)$$

which is faster than a multiply but may give less accuracy (extra roundoff). The argument is to be given in FPR0 and the answer is returned in FPR0. GPR0, GPR1, FPR0, FPR2, and FPR4 are used.

SQRTD	LTDR	2, 0	Test argument.
	BC	12, EXCP	Arg. 0 or negative.
	STE	0, TEMP	
	SR	0, 0	Zero GPR0.
	IC	0, TEMP	Insert exponent byte in GPR0.
	SRL	0, 1	Divide exponent by 2.
	A	0, MSKS1	E/2 Regenerate excess-64 characteristic.
	SR	1, 1	Zero GPR1.
	TM	TEMP, X'01'	Test exponent for even/odd.
	BC	8, EVEN	Branch to even if exponent even.

ODD	L	1, FOUR	If exponent odd, set index to 4.
EVEN	STC	0, TEMP	Insert $\left[\frac{E}{2} \right]$
	STC	0, C2(1)	Insert $\left[\frac{E}{2} \right]$
	LE	0, TEMP	Load $f \left[\frac{E}{2} \right]$
	ME	0, C1(1)	Multiply by either 8/9 or 32/9.
	AE	0, C2(1)	Initial approximation.
	LER	4, 2	Start of 1st iteration.
	DER	4, 0	x/Y_n
	AER	0, 4	$(Y_n + x/Y_n)$
	HER	0, 0	$1/2 (Y_n + x/Y_n)$
	LER	4, 2	Start of 2nd iteration.
	DER	4, 0	
	AER	0, 4	
	HER	0, 0	
	LDR	4, 2	Start of 3rd iteration-double precision.
	DDR	4, 0	
	ADR	0, 4	
	HDR	0, 0	
	DDR	2, 0	Start of 4th iteration.
	ADR	0, 2	
	MD	0, HALF	Final result.
EXCP	B	0(14)	Normal return.
	BC	4, ERROR	Error return.
	B	0(14)	Normal return for zero arg.
TEMP	DS	1E	
MSKS1	DC	X'00000020'	
FOUR	DC	X'00000004'	
C2	DC	E'.222'	2/9
	DC	E'.8889'	8/9
C1	DC	E'.8889'	8/9
	DC	E'3.55556'	32/9
HALF	DC	D'.5'	

Comments

DC type D is used for setting up long floating-point constants. The constant is aligned at the proper doubleword boundary (A.M., p. 45).

2.6 MATRIX ADDITION

Two NxM matrices, whose elements are single-length (32-bit) floating-point numbers stored row-wise starting at AMTX and BMTX respectively, are to be added and the resultant matrix is to be stored row-wise starting at RMTX.

Method 1

	LM	1, 5, REGIS	Load addresses.
	SR	1, 3	Form difference so that when added to GPR3 the correct AMTX element address is formed.
	SR	2, 3	Form difference for BMTX (see above).
LOOP	LE	0, 0(1, 3)	Load AMTX element.
	AE	0, 0(2, 3)	Add BMTX element.
	STE	0, 0(3)	Store in RMTX.
	BXLE	3, 4, LOOP	
	SVC	0	
REGIS	DC	A(AMTX, BMTX, RMTX, 4, RMTX+4(M*N-1)	
N	EQU	3	N=3 for this example.
M	EQU	5	M=5 for this example.

Comments

- Requires three registers for indexing but makes no restrictions on N and M.
- EQU is used to define a symbol (A.M., p. 36).
- $RMTX+4*(M*N-1)$ is the address of the last element on RMTX.

Method 2

	L	5, LIMIT	
	SR	13, 13	
	LA	1, AMTX	Address of AMTX to GPR1
	LA	2, BMTX	Address of BMTX to GPR2
	LA	3, RMTX	Address of RMTX to GPR3
LOOP	LE	0, 0(13, 1)	
	AE	0, 0(13, 2)	
	STE	0, 0(13, 3)	
	BXLE	13, 4, LOOP	
	SVC	0	
LIMIT	DC	A(4*(M*N-1))	

where N and M would be specified and GPR13 would take on values 0 to $4*(M*N-1)$ in increments of 4.

Comments

- Requires four registers for indexing.

Method 3

LOOP	LE	0, AMTX(13)
	AE	0, BMTX(13)
	STE	0, RMTX(13)
	BXLE	13, 4, LOOP

Comments

- Uses only one register for indexing but may not work when the matrices are large.
- Another method would be to intersperse the matrices.

2.7 PRODUCT OF TWO SQUARE MATRICES

Two NXN single-precision (32-bit) floating-point matrices L and R are stored row-wise beginning at LMTX and RMTX respectively. Create $P=L*R$ and store it row-wise beginning at PMTX. (Note: C(12) means the contents of register 12.)

	LM	6, 14, REGIS	
*			C(12) = Starting address of a row of
*			LMTX.
*			C(13) = Address of RMTX.
LOOP 2	LR	3, 13	C(3) = Index down columns of RMTX.
	LR	2, 12	C(2) = Index across rows of LMTX.
	SDR	0, 0	C(FPRO) = Sum of products.
LOOP 1	LE	2, 0(2)	Start of inner loop.
	ME	2, 0(3)	Form product of elements.
	AER	0, 2	Running sum.
	AR	3, 8	Step to next row.
	BXLE	2, 6 LOOP1	C(2) = C(2)+4 and loop.
	STE	0, 0(14)	Store element.
	AR	14, 6	C(14) = C(14)+4.
	BXLE	13, 10, LOOP2	C(13) = C(13)+4 and loop.
	SR	13, 8	C(13) = address of RMTX.
	AR	7, 8	Move to next row of LMTX.
	BXLE	12, 8, LOOP2	C(12)=C(12)+4*N and loop.
	SVC	0	
N	EQU	1000	Any positive integer.
ROW	EQU	4*(N-1)	
SQMX	EQU	4*(N*N-1)	Row +4
REGIS	DC	A(4, LMTX + ROW, ROW + 4 LMTX + SQMX)	
	DC	A(4, RMTX+ROW, LMTX, RMTX, PMTX)	

Comments

- An asterisk in column 1 indicates a comments card (A. M., p. 11).

3. COMPLETE PROBLEMS

3.1 SORTING ON THE BASIS OF SUBFIELDS (see comments)

Given 256 consecutive fields each having an "A" subfield of eight bits and a "B" subfield of 16 bits, sort on the basis of "A" at ANS. (All "A" fields are different.)

	A (8 bits)	B (16 bits)	
	ORG	4096	See comment a.
BEGIN	BALR	1, 0	
	USING	*, 1	See comment b.
	LM	4, 7, REGIS	
LOOP	LM	9, 11, 0(6)	Process four fields at a time.
	SR	12, 12	Clear GPR12.
	IC	12, 9(, 6)	Fetch "A" field into GPR12 (note displacement of nine bytes).
	AR	12, 12	Double contents of GPR12.
	STH	11, 0(12, 7)	Store "B" field (two bytes).
	SRDL	10, 24	Shift to get B at any even boundary
	SR	12, 12	
	IC	12, 6(, 6)	
	AR	12, 12	
	STH	11, 0(12, 7)	Store second field.
	SRDL	10, 24	
	SR	12, 12	
	IC	12, 3(, 6)	
	AR	12, 12	
	STH	11, 0(12, 7)	Store third field.
	SRL	9, 8	
	SR	12, 12	
	IC	12, 0(, 6)	
	AR	12, 12	
	STH	9, 0(12, 7)	Store fourth field.
	BXLE	6, 4, LOOP	Load to process four more fields.
	SVC	0	
ANS	DS	0F	See comment d.
	DS	256H	
REGIS	DC	A(12, DATA+756, DATA, ANS)	
DATA	DC	X'.... 256 data fields	
	END, BEGIN		See comment c.

Comments

- a. ORG is a pseudo instruction defining the beginning address of the program.
- b. Assignment of base registers is done automatically by the assembler, but the programmer must:
 1. Specify what registers may be used as base registers and inform the assembler of their contents
 2. Load the base registers with the appropriate values

The standard method of achieving this is through the sequence

```
BALR          R, 0
USING*, R     *, R           Where R is a register.
```

- c. END BEGIN is a pseudo instruction defining the end of the program. It also tells the monitor to start execution at BEGIN when assembly is completed.
- d. Word alignment on the fullword boundary can be achieved through a DS 0F pseudo instruction (A.M., p.51).
- e. This problem shows how information whose field length is not a fullword multiple can be handled. The reader is urged to work through this problem, showing the contents of the register at each step.

3.2 REMOVAL OF KEYWORDS

Given a string of 100 eight-bit bytes at DATA to DATA+99 (25 fullwords). Remove the words which match the four-letter keyword KEY and place the condensed result at ANS.

Method 1: Character-by-Character Operation

	ORG	4096	
START	BALR	15, 0	
	USING	*, 15	
	LA	3, DATA	Address of data to GPR3.
	LA	4, ANS	Address of answer to GPR4.
	L	0, ONE	
	L	1, NINE6	
	AR	1, 3	Ending address of data field.
	L	6, KEY	
	L	9, DATA	First four characters.
	B	CMP	
EQUAL	A	3, FOUR	
	CR	3, 1	
	BH	HIGH	Exit if data exhausted.
LODE	IC	9, 0(, 3)	Load one byte.

	SLL	9, 8	Shift eight positions to make room.
	IC	9, 1(, 3)	Load another byte.
	SLL	9, 8	Shift eight positions.
	IC	9, 2(, 3)	Load
	SLL	9, 8	Shift
	IC	9, 3(, 3)	Load
	B	CMP	
IC	IC	9, 3(, 3)	Load another byte.
CMP	CLR	6, 9	Compare four bytes to KEY.
	BE	EQUAL	Branch if equal.
	SLDL	8, 8	Shift out one byte.
	STC	8, 0(, 4)	Store the byte.
	AR	4, 0	Step answer field pointer by 1.
	BXLE	3, 0, IC	Increment GPR3 by 1.
HIGH	A	1, FOUR	
	SR	1, 3	
	BZ	END	
	STC	1, MVC+1	
MVC	MVC	0, (1, 4), 0(3)	Move the last bytes.
END	SVC	0	
ONE	DC	F'1'	
FOUR	DC	F'4'	
NINE6	DC	F'96'	
ANSW	DS	25F	
KEY	DC	C'ABC '	Keyword (note that last character is a blank).
DATA	DC	C'ABC this ABC is	
	END	START	

Comments

The MVC instruction has an SS format. The assembler language format for these instructions differs from previously encountered format (A.M., p. 24).

The instructions between LODE and the subsequent branch could have been replaced by

```

LODE    MVC        LOC, 0(3)
        L          9, LOC
        B          CMP
        -
        -
        -
LOC     DC          F'0'
```

Method 2

Use the TRT instruction to detect the occurrence of leading character in KEY, then examine next three characters for matchedness.

START	ORG	4096																				
	BALR	15,0																				
	USING	*,15																				
	LM	1,6,HOSKP	<table border="0"> <tr> <td rowspan="6">}</td> <td>INITIALIZE</td> <td>R1</td> <td>START ADDR & END OF HIT (START-1).</td> </tr> <tr> <td>WORKING</td> <td>R2</td> <td>FUNCT=2.</td> </tr> <tr> <td>REGISTERS</td> <td>R3</td> <td>START+98.</td> </tr> <tr> <td></td> <td>R4</td> <td>0, NEW START ADDR WORK REG.</td> </tr> <tr> <td></td> <td>R5</td> <td>0, LENGTH OF REMAINING SCAN.</td> </tr> <tr> <td></td> <td>R6</td> <td>ANSWER ADDRESS.</td> </tr> </table>	}	INITIALIZE	R1	START ADDR & END OF HIT (START-1).	WORKING	R2	FUNCT=2.	REGISTERS	R3	START+98.		R4	0, NEW START ADDR WORK REG.		R5	0, LENGTH OF REMAINING SCAN.		R6	ANSWER ADDRESS.
}	INITIALIZE	R1	START ADDR & END OF HIT (START-1).																			
	WORKING	R2	FUNCT=2.																			
	REGISTERS	R3	START+98.																			
		R4	0, NEW START ADDR WORK REG.																			
		R5	0, LENGTH OF REMAINING SCAN.																			
		R6	ANSWER ADDRESS.																			
LOOP2	LR	4,1	New start addr → R4.																			
LOOP1	LR	5,3	End addr → R5.																			
	SR	5,1	Length of TRT.																			
	EX	5,TRT	Execute TRT inst.																			
	BC	5,OUT	End of field.																			
	CLC	KEY+1(3),1(1)	Test remainder of key.																			
	BNE	LOOP1	Continue test cycle on ≠.																			
	SR	1,4	} Set up length for data move to answer area.																			
	SR	1,2																				
	EX	1,MVC	Execute data move.																			
	LA	6,1(1,6)	} Set up to test remainder of the field.																			
	LA	1,5(1,4)																				
	BC	LOOP2	Continue test.																			
OUT	SR	3,4	Calc. final length.																			
	EX	3, MVC	Execute final move.																			
	SVC	0	Return to monitor.																			
TRT	TRT	1(0,1),TABLE																				
MVC	MVC	0(0,6),1(4)																				
HOSKP	DC	(Storage for 6 reg. loading)																				
TABLE	DC	(Table for TRT inst (Function=2))																				

PRIME NUMBER GENERATOR

Find all prime numbers whose value is less than a certain number LIM. Algorithm: To decide whether a number N is a prime, we divide it by all primes with value $\leq \sqrt{N}$. If no exact divisor is found, N is a prime.

	ORG	4096	
START	BALR	15,0	
	USING	*,15	
	L	14,ONE	
	ST	14,M	First number = 1.
	AR	14,14	
	ST	14,M+4	Second prime = 2.
	A	14,ONE	

	ST	14, M+8	Third prime = 3.
	SR	13, 13	Zero GPR13.
	ST	13, MNS	Zero MNS
	ST	14, NM	Store C(14) in previous prime number.
	L	9, TWELVE	GPR9 = Index on prime number array.
LOOP1	L	14, NM	Load previous number.
	A	14, M+4	Add 2 to previous number.
	ST	14, NM	Next number to be tested.
	S	14, LIM	Test for end.
	BP	STOP	
	L	13, EIGHT	
LOOP2	LM	10, 11, MNS	Load current numbers to be tested.
	D	10, M(13)	Remainder in Reg. 10.
	BXLE	10, 11, LOOP1	If Rem = 0, number is not prime, so branch to LOOP1 to next candidate.
	S	11, M(13)	Square root test.
	BM	PRIME	Number is prime.
	A	13, FOUR	Try next prime number as divisor.
	B	LOOP2	
PRIME	L	8, NM	Load number found to be prime.
	ST	8, M(9)	Store in prime array.
	A	9, FOUR	Bump prime array index.
	B	LOOP1	
STOP	SVC	0	
MNS	DS	1F	
NM	DS	1F	Previous prime number.
ONE	DC	X'00000001'	
FOUR	DC	X'00000004'	
EIGHT	DC	X'00000008'	
TWELVE	DC	X'0000000C'	
LIM	DC	X'00000064'	LIM = 100 in this example.
M	DS	50E	Prime numbers stored here.
	END	START	

3.4 MIN-MAX PROBLEM

Given an array of N rows and M columns, whose elements are 32-bit integers stored row-wise beginning at ARRAY. Find the maximum of each row and store the addresses of these maxima in fullwords beginning at MAX. Then find the minimum of these maxima and store its address at MIN. For this example an array of three rows and five columns is used.

	ORG	4096	
START	BALR	15, 0	
	USING	*, 15	
	LM	2, 7, REGIS	
	L	1, 0(3)	C(1) = MIN of MAX.
	ST	3, MIN	Address of current minimum.
REPLACE	L	14, 0(3)	C(14) = MAX.

	ST	3, MAX(2)	Address of current maximum for this row.
	B	BXH	
CMP	C	14, 0(3)	Compare current maximum with operand new
	BL	REPLACE	maximum discovered.
BXH	BXH	3, 4, CMP	Repeat until row is exhausted.
	CR	1, 14	Is NEW MAX less than MIN?
	BH	NEWMIN	Branch if so.
AR	AR	2, 4	Handle next member of MAX vector.
	BXH	5, 6, REPLACE	Proceed to next row.
	SVC	0	
NEWMIN	LR	1, 14	New minimum generated.
	L	10, MAX(2)	
	ST	10, MIN	Address of new MIN saved.
	B	AR	
N	EQU	3	
M	EQU	5	
END	EQU	4*(N*M-1)	
MT4	EQU	M*4	
ARRAY	DC	F'1, 3, 5, 7, -9, -1, -3, -50, -7, 8, 2, 4, 0, 8, 10'	
MAX	DS	5E	
MIN	DS	1E	
REGIS	DC	A(4*N-4, ARRAY+END, -4, ARRAY+END-MT4, -MT4, ARRAY-8)	
	END	START	

Comments

- a. Candidate for max is at GPR14, with address of candidate at MAX(2). Candidate for min is at 1, with address at MIN.

3.5 DOUBLE-PRECISION BINARY-TO-DECIMAL CONVERSION

Given a 64-bit binary positive integer beginning at fullword boundary BIN, convert it into decimal beginning at DEC.

	ORG	4096	
START	BALR	15, 0	
	USING	*, 15	
	LM	2, 3, BIN	64-bit number to GPR2 and GPR3.
	CVD	2, DEC+8	Convert high-order part.
	MP	DEC(16), TWO32(6)	Decimal-multiply high-order by 2^{32} .
	SLDA	2, 31	Shift low order to GPR2. Save low-order bit in sign of GPR3.
	CVD	2, TEMP	Convert low-order part.
	MP	TEMP(8), TWO(1)	Decimal-multiply low order by 2.
	LTR	3, 3	Set condition code.
	BC	4, ADD	Test for low-order bit.
CONT	AP	DEC(16), TEMP(8)	Add two parts together.
	SVC	0	
ADD	AP	TEMP(8), ONE(1)	Add in low-order one.
	B	CONT	

TWO32	DC	X'04294967296C'	Decimal constant 2^{32} .
TWO	DC	X'2C'	Decimal constant 2^1 .
ONE	DC	X'1C'	Decimal constant 1.
TEMP	DS	1D	
DEC	DS	2D	
	END	START	

Comments

- A 63-bit binary integer plus sign is equivalent to a decimal number of not more than 20 digits plus sign. A 128-bit field is used to store this decimal result.
- The CVD instruction converts a 32-bit signed integer to decimal. The 63-bit magnitude field is, therefore, broken into two 31-bit fields and a 1-bit field. The high-order part is converted and then multiplied by 2^{32} ; the lower part is converted and multiplied by 2. The remaining bit contributes either a 0 or 1 to the result.

3.6 SCANNING A MESSAGE FOR SYNTACTICAL ERRORS

SIMSCAN is a very simple scanner which could be used for precompilation scanning of programs. In this use it would locate low-level syntactic errors without wasting the time of the compiler proper.

The code shows a standard use of TRT (translate and test). It also exemplifies use of several general registers for keeping track of a multidimensional situation.

Problem

Given a string of characters starting at a known location and terminating with a period. The string is called error-free if and only if:

- There is either one or no equal sign.
- No two arithmetic connectives (+, -, *, /) are adjacent.
- At no point during a left-to-right scan have more right parentheses than left parentheses been encountered, and the total number of occurrences of each is the same.

Solution

SIMSCAN is written as a quasi-subroutine. The program below contains not only SIMSCAN but a main program called CONTROL and several input strings.

The register usage table, which follows, explains the code:

General Purpose		Contents
Register No.		
0		= 1
1		Argument address for TRT.
2		Function byte for TRT.
3		(not used)
4		"
5		"
6		"
7		= 256
8		Left-right parenthesis count (start at 0).
9		Equal sign count (start at -1).
10		Address of last previous connective encountered.
11		Base address for STRINGS.
12		Base address for SIMSCAN.
13		= 4
14		Length of current statement.
15		Program base address.
CONTROL	CSECT	Controlling program.
	BALR 15, 0	Addressability.
	USING *, 15	
	L 7, TWO56	
	L 12, ADSCAN	Load bases.
	L 11, ADSTRNGS	
	L 14, 0(11)	Length of current string.
	L 13, FOUR	= 4
	L 0, ONE	= 1
	SR 2, 2	
	AR 11, 13	Skip length word.
	LR 1, 11	Input to simscan.
	BR 12	Call simscan.
NEWDATA	AR 11, 14	Address of new input.
	L 14, 0(11)	
	LTR 14, 14	Check for end signal
	BM FINISHED	
	AR 11, 13	
	LR 1, 11	
	BR 12	
FINISHED	SVC 0	Stop
ERROR	SVC 1	Print
	DC X'0010'	
	B NEWDATA	
ADSCAN	DC A(SCAN)	
ADSTRNGS	DC A(STRINGS)	
FOUR	DC F'4'	

ONE	DC	F'1'	
TWO56	DC	F'256'	
SIMSCAN	CSECT		* * * Scanner * * *
	USING	*,12	Loaded by control.
	SR	8,8	Clear scratch pad.
	LCR	9,0	Clear scratch pad.
	LCR	10,13	Clear scratch pad.
	SR	1.0	Initialize.
CONTINUE	AR	1.0	Advance beyond detected symbol.
	TRT	0(255.1), TABLE	Start or continue scan.
	BC	6,SIMSCAN(2)	Symbol found, take action.
	AR	1,7	No symbol, advance,
	B	CONTINUE	Continue.
LPAREN	AR	8,0	Increase (count.
	B	CONTINUE	
RPAREN	SR	8,0	Decrease (count.
	BM	ERROR	
	B	CONTINUE	
EQUALS	AR	9,0	Increase = count.
	BP	ERROR	Check for .GT. 1.
	B	CONTINUE	
CONNECT	AR	10,0	See if last previous
	SR	10,1	connective
	BC	10,ERROR	was too close.
	LR	10,1	Reset
	B	CONTINUE	
PERIOD	LTR	8,8	End of message
	BP	ERROR	Check (count.
	B	NEWDATA	
TABLE	DC	9D'0'	for ASCII code
	DC	AL1(LPAREN-SIMSCAN)	
	DC	AL1(RPAREN-SIMSCAN)	
	DC	AL1(CONNECT-SIMSCAN)	
	DC	AL1(CONNECT-SIMSCAN)	
	DC	2X'0'	
	DC	AL1(CONNECT-SIMSCAN)	
	DC	AL1(PERIOD-SIMSCAN)	
	DC	AL1(CONNECT-SIMSCAN)	
	DC	13X'0'	
	DC	AL1(EQUALS-SIMSCAN)	
	DC	X'00'	
	DC	20D'0'	
STRINGS	CSECT		
	DC	F'30'	
	DC	C'THIS (E*F-L+-). . . .	'
	DC	F'31'	
	DC	C'BEEP (())(()))(('
	DC	F'10'	
	DC	C'X=Y+Z=2	'
	DC	F'10'	

	DC	C'((X)((.	'
	DC	F'10'	
	DC	C'X+Y(=)-2Z.'	
	DC	F'-1'	
TABLE	DC	9D'0'	for EBCDIC code
	DC	CL3'000'	
	DC	AL1(PERIOD-SIMSCAN)	
	DC	CL1'0'	
	DC	AL1(LPAREN-SIMSCAN)	
	DC	AL1(CONNECT-SIMSCAN)	
	DC	18CL1'0'	
	DC	AL1(CONNECT-SIMSCAN)	
	DC	12CL1'0'	
	DC	AL1(CONNECT-SIMSCAN)	
	DC	15CL1'0'	
	DC	AL1(EQUALS-SIMSCAN)	
	DC	CL1'0'	
	DC	16D'0'	



International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York 10601