A Study of the Round Operation on the IBM System/360

by

John H. Welsch

SLAC - Stanford University

August 1967

Floating-point arithmetic and type conversion operations are all truncated rather than rounded on the IBM 360. This report discusses rounding operations using existing instructions and proposed new instructions.

Introduction

There are no rounded operations on the IBM System/360. All floating-point instructions, type conversion (FIX and FLOAT), and precision conversion (long to short form) truncate (chop) the result to the proper word length. There are many cases when this conversion should be done with minimum error by rounding rather than truncating. In this report we present three approaches to rounding long precision floating-point numbers to short precision.

Approach A: using existing instructions,

Approach B: using several special added instructions,

Approach C: by a round instruction.

Rounding

Rounding a long precision (14 hexadecimal digit fraction) floating-point number to a short precision (6 hex digits) floating-point number may be done by adding a hexadecimal digit 8 in the seventh hex digit position to the magnitude of the long precision fraction and truncating the fraction to 6 hex digits. Similarly, the digit in the seventh hex position may be doubled and the carry, if any, propagated left. The rounding operations described below do not prenormalize the fraction but do postnormalize by a one digit right shift if necessary before truncation (which may cause exponent overflow).

New Instructions

The coding for the statement X = RND(Y), for Y long precision and X short precision is shown for all three approaches in Figure 1. It is assumed that a compiler would emit in-line code as for ABS( ) and SIGN ( ). The two additional instructions used in approach B are:

LDGR R_1,R_2 [RR,Long Operands]

the contents of the even-odd general register pair specified by the second operand location are placed in the floating-point register R_1,

STGR R_1,R_2 [RR,Long Operands]

the contents of floating-point register R are placed in the even-odd general register pair specified by the second operand.

The times for these two instructions are taken to be the same as for the LDR instruction on the various models.

Approach C assumes the existence of a micro-programmed or hardware round instruction RDR described in Figure 2. The execution time is taken to be the same as for the ADR instruction on the various models. A summary of the timing estimates is given in Table 1.

	Approach				
times in μ seconds	A	B/A%	B	C/A%	C
Model 50	38.67	77%	29.90	48%	18.84
Model 65I	10.02	84	8.44	40	4.05
Model 75I	6.97	56	3.94	34	2.40
Length (bytes)	30	73%	22	33%	10

Timing Estimates

Table 1

Conclusions

People wishing to fully utilize short precision floating-point representation would do well to evaluate all arithmetic expressions using long precision floating-point arithmetic with rounding to short precision at the appropriate places. The cost of approach C just for rounding might be too high to justify, but approach B uses two instructions which would find many other applications, and if compilers used them to advantage, there would be a definite improvement in program performance.

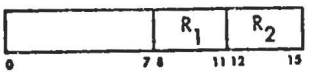
Approach	Operation	Operands	Comments	Length	50	65I	75I
A							
ROUND	LD	0,Y		4	6.00	1.40	.70
	STD	0,TEMP		4	6.00	.93	.82
	XC	TEMP+1(3),TEMP+1	Zero 3 bytes	6	13.67	4.36	3.74
	AW	0,TEMP		4	9.00	2.40	.89
	STE	0,X		4	4.00	.93	.82
TEMP	DS	1D		8			
				30	38.67	10.02	6.97
				(bytes)	μ seconds		
B							
ROUND	LD	0,Y		4	6.00	1.40	.70
	STGR	0,0	FPR0 to R0 and R1	2	3.50	1.23	.40
	N	0,MASK	Zero 3 fraction bytes	4	5.75	2.00	.77
	LDGR	2,0	R0 and R1 to FPR2	2	3.50	1.23	.40
	AWR	0,2		2	7.15	1.65	.85
	STE	0,X		4	4.00	.93	.82
	DS	0F					
MASK	DC	X'FF000000'		4			
				22	29.90	8.44	3.94
				(bytes)	μ seconds		
C							
ROUND	LD	0,Y		4	6.00	1.40	.70
	RDR	0,0	Round	2	7.84	1.72	.85
	STE	0,X		4	4.00	.93	.85
				10	18.84	4.05	2.40
				(bytes)	μ seconds		

Timing Details
Figure 1

Figure 2

Round

RDR R_1, R_2 [RR, Long Operands]



The long operand in the second register is placed, rounded to short precision, in the first operand location.

The high-order half of the second register is transferred to the high-order half of the first register and the low-order half of the first register is set to zero. If bit position 32 of the second operand was one, a one is added at position 31 to the magnitude of the fraction placed in the first operand location. If an overflow carry occurs, the fraction is right-shifted one digit, and the characteristic is increased by one. If this increase causes a characteristic overflow, an exponent-overflow exception is signaled, and a program interruption occurs.

Condition Code: The condition code remains unchanged.

Program Interruptions:

Operation (if floating-point feature is not installed)
Specification
Exponent overflow