The S/390 G5 Floating Point Unit Supporting Hex and Binary Architectures

Eric M. Schwarz, Ronald M. Smith, Christopher A. Krygowski
S/390 Microprocessor Development
IBM System 390 Division
522 South Rd., MS:P310
Poughkeepsie, NY 12601

Abstract

The first high performance floating point unit to support both IBM 360 hexadecimal based floating point architecture and the IEEE 754 Standard binary floating point architecture is described. The S/390 G5 floating point unit supports the new S/390 architecture which includes hexadecimal based short, long, and extended precision formats and IEEE 754 standard single, double, and quad formats. This floating point unit is part of the microprocessor chip on the S/390 G5 mainframe computer introduced in 1998 and generally available at 500 MHz speeds. The S/390 G5 represents the current state of the art in CISC processor design. This paper describes the S/390 architecture enhancements, the internal format of the FPU, and the modifications to the FPU dataflow.

1. Introduction

S/390 mainframes use a proprietary floating point format dating back to the 360 architecture introduced in the 1960s[1]. The format has hexadecimal exponents and hex digit normalization. It was created with the strong influence of early hardware limitations. In the SSI technology of 1960s, shifting the mantissa was a major operation. The 360 architecture reduces the necessary shifting by defining normalization to a hex digit boundary rather than a binary bit boundary. The exponent can be represented with two less bits than a binary based format which allows the sign and exponent to fit into one byte. The resulting mantissa format can have up to 3 leading zero bits in a normalized number. Floating point addition is defined to perform alignment based on exponents regardless of normalization, and to maintain one hex guard digit during the calculation of an intermediate result. Hexadecimal floating point architecture is designed to allow simple and fast implementations.

The IEEE 754 standard was developed in 1985 [2] to standardize computation among several manufacturers and to enforce a mathematically "pure" result. The format is very efficient and numbers are binary normalized with the leading one implied rather than explicitly represented. The range of exponents varies between formats and allows a larger range than the hex format. The IEEE 754 standard has been adopted almost universally in the PC, workstation, and midrange computer markets. S/390 mainframes have been using an incompatible floating point format until 1998.

To expand the markets in which S/390 mainframes compete, the architecture has been expanded to integrate the two floating point architectures. Both the old hexadecimal based architecture and the relatively new IEEE 754 binary based architecture are supported. The first machine to implement this enhancement is the S/390 G5 processor announced May 7, 1998 and generally available in September 1998. The implementation provides compatibility to both architectures but is optimized to hexadecimal architecture.

1.1. Architecture

This section describes the architectural features of the new ESA/390 Binary Floating Point facility[5]. Formats, registers, instructions, rounding modes, and exceptions are described.

1.1.1 Formats

The ESA/390 architecture defines 6 floating point formats as described in Table 1. The S/390 formats are based on the S/360 architecture developed in the 1960s. To distinguish between the two formats now available

Format	Sign (s)	Characteristic(c)		Mantissa(m)	Total Width
	bits	bits	bias	bits	bits
Hex Short	1	7	64	24	32
Hex Long	1	7	64	56	64
Hex Extended	1	7	64	112	128
Binary Short	1	8	127	24	32
Binary Long	1	11	1023	53	64
Binary Extended	1	15	16383	113	128

Table 1. Supported Formats

in S/390 the old hexadecimal format is referred to as HFP and the binary format is referred to as BFP. An HFP number can be expressed by the following equation:

$$X_{hex} = (-1)^{X_s} * 0.X_m * 16^{X_{Cha}-bias}$$

where X_{hex} is value of the hex format number, X_s is the sign bit, X_m is the mantissa which is less than 1.0, X_{Cha} is the characteristic (hex architected format), and the bias is fixed at 64. There are 3 HFP formats. They all have 7 bit characteristics, but the size of the mantissa varies from 24 to 112 bits. The extended format is equivalent to two 64 bit long format numbers whose second characteristic is 14 less than the first characteristic. This is similar to a double-double format with contiguous fraction bits.

A normalized BFP number can be expressed by the following equation:

$$X_{binary} = (-1)^{X_s} * 1.X_m * 2^{X_{Cba}-bias}$$

where X_{binary} is value of the binary format number, and X_{Cba} is the characteristic (binary architected format). S/390 defines BFP short and long formats which are equivalent to the IEEE 754 single and double formats. Additionally, a binary extended format is defined which is equivalent to IEEE 754 type quad format adopted by several manufactures such as Hewlett-Packard's PA-RISC architecture [3], SUN Microsystems's SPARC, and DEC's Alpha[4]. The Alpha architecture also supports two floating point architectures VAX and IEEE though both are binary based. All BFP formats have an implied one for normalized numbers and allow denormalized numbers.

1.1.2 Registers

With the installation of this new S/390 architecture the floating point register file expands from 4 to 16 64-bit registers. Recent S/390 machines were able to cope with this limitation by optimizing the pipeline for RX (register and memory) type instructions.

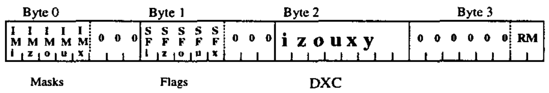

Figure 1. Floating Point Control Register

One control register is defined and is called the Floating Point Control register (FPC) and is shown in Figure 1. It is 4 bytes in length and contains the mask bits in byte 0, flag bits in byte 1, data exception code (DXC) in byte 2, and the rounding mode in byte 3. Mask and flag bits are available for invalid operation (i, bit 0), division by zero (z, bit 1), overflow (o, bit 2), underflow (u, bit 3), and inexact (x, bit 4). The other bits are reserved. The data exception code is similar for bits 0 to 4 but contains a bit 5 (y) which represents if an inexact result is incremented in magnitude. The rounding mode is stored in byte 3, bits 6 and 7. A rounding mode of "00" is round to nearest as defined by the IEEE 754 standard and sometimes called round to nearest even, "01" is round toward zero, "10" is round toward positive infinity, and "11" is round toward negative infinity. The FPC rounding mode determines the direction of rounding for numbers that can't be exactly machine represented. A few instructions include a M3 or M4 field in the instruction text which can override the current FPC specified rounding mode. This field can specify the use of the current rounding mode, force any of the 4 rounding modes specified above, or force biased round to nearest (sometimes called simple rounding). Thus, 5 rounding modes are supported in hardware.

1.1.3 Instructions

The Binary Floating Point facility adds a significant number of opcodes to S/390 architecture. A signifi-

Instruction Type	S	L	X
Add / Subtract	H/B	H/B	H/B
Add / Sub. Unnormalized	H	H	-
Compare	H/B	H/B	H/B*
Compare and Signal*	B	B	B
Convert from Fixed*	H/B	H/B	H/B
Convert to Fixed*	H/B	H/B	H/B
Divide	H/B	H/B	H/B
Divide to Integer*	B	B	-
Halve	H	H	-
Load and Test	H/B	H/B	H/B*
Load Complement	H/B	H/B	H/B*
Load FP Integer*	H/B	H/B	H/B
Load Lengthened*	H/B	H/B	H/B
Load Negative	H/B	H/B	H/B*
Load Positive	H/B	H/B	H/B*
Load Rounded	H/B	H/B	H/B*
Multiply	H/B	H/B	H/B*
Multiply and Add/Sub*	B	B	-
Square Root	H/B	H/B	H/B*
Test Data Class*	B	B	B
SUPPORT			
Convert BFP to HFP*	X	X	-
Convert HFP to BFP*	X	X	-
Load RR	X	X	X*
Load RX	X	X	-
Load Zero*	X	X	X
Store	X	X	X
CONTROL			
Extract FPC*	-	-	-
Load FPC*	-	-	-
Set FPC*	-	-	-
Set Rounding Mode*	-	-	-
Store FPC*	-	-	-

Table 2. Instruction Types

cant issue in the overall design was whether a modal approach could be used for switching between HFP and BFP. The overriding factor was that multiple behaviors for the same opcodes caused significant compatibility and linkage problems for both the operating system and application software, so the modal behavior design was dropped in favor of separate new opcodes for the BFP operations. There are 54 previously defined hex instructions, but the new Binary Floating Point facility adds 121 opcodes which consist of 8 new support instructions, 26 new hex instructions, and 87 binary instructions.

Table 2 shows a list of all the instruction types now available in S/390 architecture with the Binary Floating Point facility installed. The instructions are separated into instructions that apply to both HFP and BFP, support instructions, and control instructions. The table describes whether instructions are available in hex format (H), binary format (B), or both (H/B), and for short (S), long (L), or extended (X) precision formats. An asterix (*) denotes a new instruction to the architecture.

1.1.4 Exceptions

IEEE 754 standard defined traps are implemented as S/390 data exceptions. Each type of trap has a separate data exception code (DXC). All 5 types of exceptions as described in the IEEE 754 standard, Section 7 [2] are implemented.

Underflow exception uses the same two rules for detection as the RS/6000: tininess is detected prior to rounding and loss of accuracy is determined by the exactness of the result.

Overflow and underflow exceptions for binary format conversion to a smaller format cause strange but defined results in compliance to sections 7.3 and 7.4 of the IEEE 754 Standard[2]. The standard dictates that the mantissa be rounded to the target format on a trap. The exponent could wrap multiple times in the target format, so it is maintained in the source format. But even in the source format it could wrap the exponent due to rounding the mantissa. Therefore the exponent is wrapped with a fixed bias adjust and kept in the source format with an exponent in the target and the resulting representation is to the length of source format. This is accomplished all in hardware on the S/390 G5 FPU.

There are also two other new S/390 data exception codes for the binary floating point facility. They are related to a new control register bit (CR0.13) called the Additional Floating Point (AFP) register control bit. This facility provides a means for disabling the new function even though on the G5 machine this hardware is always considered to be installed.

1.2. Software Support

As of September 1998 $OS/390^{TM}$ Version 2 Release 6 supports binary floating point. There has been a world wide effort at pulling together operating system support, linkage editors, run time libraries, compilers, and applications development for this release. Some of the items include making operating system aware of the additional floating point registers to save and restore on context switches and be able to link the correct libraries together for HFP or BFP. User features include a C/C++ compiler and a C Run Time Library supporting BFP. Also, Java is becoming an industry standard which requires underlying IEEE 754 floating point support. Java is supported by a Just-In-Time compiler. In terms of applications support, Lotus Notes and DB2 are updated to utilize this hardware.

1.3. Hardware/Technology

The G5 microprocessor uses the IBM CMOS6X technology which has a device size of 0.25 microns drawn and 0.15 microns effective length (nFET)[6]. There are 6 metal layers and the supply voltage is 2.0 volts. The microprocessor chip is 14.6 mm by 14.7 mm and contains 25 million transistors. The chip operates at 25 Watts. The product ship frequency is 500 MHz but the chip has been tested at up to 600 MHz in a laboratory environment[9]. The processor has a performance of 150 S/390 MIPS [1] for uni-processor and 1040 S/390 MIPS for a 10-way processor which is the highest in the industry in 1998 including both CMOS and bipolar mainframes. This is over twice the performance of the S/390 G4 microprocessor (64 and 450 MIPS) from 1997. The FPU is replicated on the microprocessor chip for error tolerance and takes up approximately 10 percent of the total chip area. The G5 FPU is approximately 20% larger than the G4 FPU prior to technology scaling to accommodate the BFP architecture.

1.4. Optimization

The S/390 G5 processor operates in a commercial environment rather than in a scientific environment. Therefore, tradeoffs in area and timing over floating point performance have been made. The current expected workload of the machine is traditional applications for hex floating point but binary floating point applications are expected to grow rapidly. With this in consideration the G5 FPU is optimized for HFP but with hardware implementation and full functionality of BFP. Most instructions including extended formats and special cases are implemented in hardware. Most short and long operations are pipelined and extended precision instructions are non-pipelined. The converts to/from fixed, divide to integer, test data class, and the control instructions are the few floating point instructions implemented in low level software called millicode.

The G5 processor is based on the G4 processor design and was developed by the same team. The dataflow could not be altered extensively given the constraints of the schedule of this follow-on machine. Thus, the major dataflow functions remain the same such as the multiplier[7] and most of the basic dataflow[8]. Three binary formats and twice as many instructions were added to the design with minor modifications. The rest of this paper shows how this significant architectural revision is adapted into the G4 dataflow to provide full functionality of the BFP architecture with reasonable performance and minimal changes.

[1] S/390 MIPS are determined by a geometric average of many commercial workloads.

2. Internal Architecture

The six floating point formats are supported in the floating point unit by using one internal format. The input operands are transformed into this internal format and all computation is performed on this format and then the intermediate result is transformed into the output architected format. To optimize for the HFP architecture the internal format is chosen to be a hex based format. This makes the conversion between the HFP formats and the internal format trivial, but slightly hinders the BFP format.

The internal format is hexadecimal based with a 14 bit characteristic and a bias of 8192 as expressed by the following:

$$X_{internal} = (-1)^{X_s} * 0.X_m * 16^{X_{Chi}-8192}$$

where $X_{internal}$ is value of the internal format number, and X_{Chi} is the characteristic (hex internal format). The hex architected 7 bit characteristic is easily supported by the 14 bit format. BFP format uses at most 15 bits of characteristic but with a binary exponentiation. A direct comparison between binary and hex exponentiation is that 2 less bits are required for hex as shown by the following relation:

$$2^{4X} = 16^{X}$$

since 4X contains two more bits than X.

The 1997 S/390 G4 FPU actually has a 14 bit exponent dataflow which was implemented in preparation for the G5 machine and the addition of BFP architecture. In the G4 FPU the additional exponent bits were not used since it only supported HFP architecture, but this enabled the dataflow stack bit width to remain constant between machines.

3. Dataflow Modifications

The S/390 G4 and G5 FPUs have a 56 bit fraction dataflow with a floating point multiplier which uses a radix-8 algorithm[7]. There is a 3X adder and Booth decode in the first cycle of execution, Booth multiplexing and a 19 to 2 counter tree in the second cycle, and a 120 bit adder in the third cycle. The FPUs also have a floating point addition pipeline which shares the 120 bit adder with the multiplier. Alignment and complementation is computed in the first cycle, the 120 bit

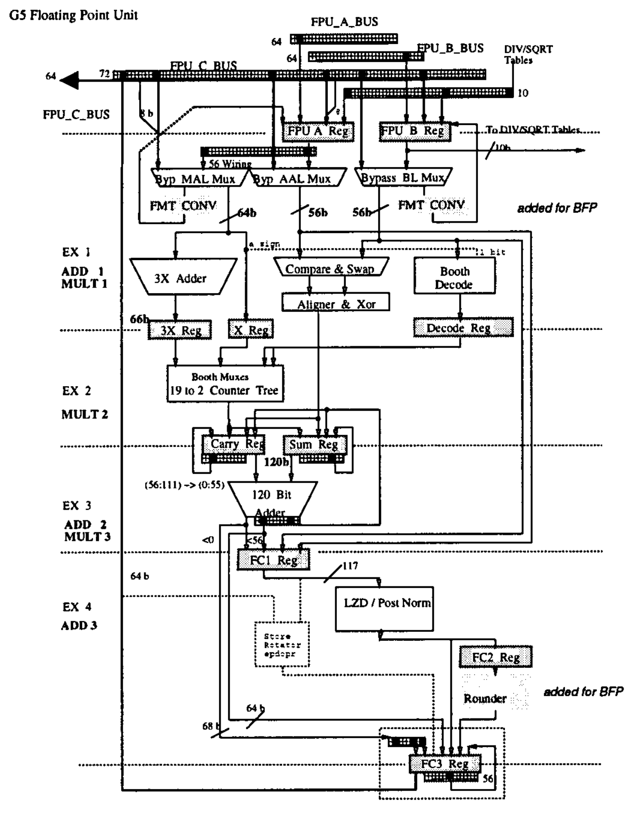

Figure 2. G5 FPU Fraction Dataflow

addition in the second cycle, and post normalization in the third cycle. Most HFP multiplication and additions have a latency of 3 cycles and have a throughput of 1 every cycle.

The G5 FPU fraction dataflow is shown in Figure 2. The G5 adds format converters below the A and B registers for converting binary architected format operands into hex internal format and detecting special input numbers. Also, sticky bit detectors are added to the aligner and normalizer. And, a rounder / format converter is added at the bottom of the pipeline to complete the IEEE rounding and to convert back to binary architected format. Each of these units will be described in more detail.

3.1. Conversion from Hex Architected to Hex Internal and Back

The conversion between hex architected and hex internal format is trivial. The fraction and sign bits are the same in the two formats. So, only the characteristic is converted. The 7 bit characteristic in the architected format is transformed into a 14 bit characteristic by something similar to sign extension as shown by the following:

$$\begin{aligned} 16^{(X_{Chi}-8192)} &= 16^{(X_{Cha}-64)} \\ X_{Chi} - 8192 &= X_{Cha} - 64 \\ X_{Chi} &= X_{Cha} + 2^{13} - 2^{6} \end{aligned}$$

Note that $2^{13} - 2^{6}$ represents a string of ones from the next to most significant bit of the 14 bit internal format to the most significant bit of the architected characteristic. This constant addition does not effect the low 6 bits of the architected characteristic and results in an extension of the most significant bit followed by 7 instances of its complement as shown by the following:

$$\begin{aligned} X_{Chi} &= 0000000C_0C_1C_2C_3C_4C_5C_6 + \\ &\quad 01111111000000_2 \\ &= C_0\overline{C_0}\,\overline{C_0}\,\overline{C_0}\,\overline{C_0}\,\overline{C_0}\,\overline{C_0}\,\overline{C_0}C_1C_2C_3C_4C_5C_6 \end{aligned}$$

The internal format is chosen specifically to optimize the transformation of hex input operands and this sign extension of the characteristic is performed while loading the operands into the A and B input registers. The internal bias is a power of two (i.e. $2^{13} = 8192$) which is hex-like versus the binary format bias which is a power of two minus one (i.e. $2^{15} - 1 = 16383$).

After the intermediate result in the hex internal format is calculated it is transformed back into the architected format so it can be written into the register file. The register file contains the numbers in architected format. The transformation from hex internal format to hex architected format is simply the reverse of the sign extension. The least significant 6 bits are preserved and the 7th bit is inverted which is preferred over transmitting the most significant bit of the internal characteristic. If the characteristic is within the exponent range of the architected format the most significant bits are guaranteed to be $C_0\overline{C_0}\,\overline{C_0}\,\overline{C_0}\,\overline{C_0}\,\overline{C_0}\,\overline{C_0}\,\overline{C_0}$. Hex architecture dictates on overflow and underflow exception that the characteristic wrap from 127 to 0 or 0 to 127. The internal characteristic is in a wider precision and this is automatically what is occuring to bit 7. So, inverting bit 7 for either the normal or the exception case provides the most significant bit of characteristic.

Note both format conversion to and from hex architected requires very little hardware and computation, so it is accomplished while loading or unloading operands from the FPU.

3.2. Conversion from Binary Architected to Hex Internal

The conversion from binary architected format to hex internal is a little more complex since both the fraction and characteristic need to be modified. A transformation of exponents between the two formats is as simple as dividing the base two exponent by 4 (shifting by 2) and applying the residual to a binary shift of the fraction. The resulting fraction requires 53 bits plus up to 3 bits of leading zeros to represent in a hex normalized format which is exactly 56 bits, the size of the hex long fraction. The following shows this transformation if exponents are considered rather than characteristics and their biases:

$$\begin{aligned}
1.X_m * 2^0 * 2^E &\Rightarrow (0.0001||52bits) * 16^{(E>>2)+1} \\
1.X_m * 2^1 * 2^E &\Rightarrow (0.001||52bits||0) * 16^{(E>>2)+1} \\
1.X_m * 2^2 * 2^E &\Rightarrow (0.01||52bits||00) * 16^{(E>>2)+1} \\
1.X_m * 2^3 * 2^E &\Rightarrow (0.1||52bits||000) * 16^{(E>>2)+1}
\end{aligned}$$

where E is divisible by 4.

The conversion is more difficult since the biases differ for hex and binary. The following shows the formulation where $r = Cba \; mod \; 4$ and the binary bias is represented by $2^N - 1$.

$$1.X_m * 2^{Cba-bias} \Rightarrow hex\_frac * 16^{Chi-8192}$$

$$\begin{aligned}
&= 1.X_m * 2^{(Cba-(2^N-1))} \\
&= 1.X_m * 2 * 2^{(Cba-2^N)} \\
&= 1.X_m * 2 * 16^{(\frac{Cba}{4}-2^{(N-2)}+8192)-8192} \\
&= 1.X_m * 2 * 2^r * 16^{(\lfloor\frac{Cba}{4}\rfloor-2^{(N-2)}+8192)-8192} \\
&= (0.001X_m * 2^r) * 16^{(\lfloor\frac{Cba}{4}\rfloor+1+2^{13}-2^{(N-2)})-8192}
\end{aligned}$$

This function of complementing the most significant bit and extending it to bit 0 of the 14 bit notation with the most significant bit not complemented (true) is defined to be the function SE. Below is a table for these conversions:

$$1.X_m * 2^{Cba-bias} =$$

$$\begin{aligned}
For\;\; r = 3, &\Rightarrow (0.0001||52bits) * 16^{(SE(Cba>>2)+2)-8192} \\
For\;\; r = 0, &\Rightarrow (0.001||52bits||0) * 16^{(SE(Cba>>2)+1)-8192} \\
For\;\; r = 1, &\Rightarrow (0.01||52bits||00) * 16^{(SE(Cba>>2)+1)-8192} \\
For\;\; r = 2, &\Rightarrow (0.1||52bits||000) * 16^{(SE(Cba>>2)+1)-8192}
\end{aligned}$$

To accomplish this transformation two format converters are placed next to A and B register. The outputs of the A and B registers feed both the normal dataflow and the format converters and they output their results back to these registers. The format converters consist of a series of multiplexors in the fraction dataflow and multiplexors and an adder in the exponent dataflow. In the fraction dataflow first the format is multiplexed to determine which bits are exponent bits and which bits are the fraction. The multiple fraction formats are multiplexed and the implied one is added if the exponent is not all zeros. The fraction is then shifted depending on the least significant two bits of exponent by another multiplexor. The output fraction is fed back to the input fraction register and also to a special number detector which compares the fraction to zero. In the exponent format converter the first multiplexor performs a length conversion and sign extension function. Then the least significant two bits are used to select a constant of 1 or 2. Then the constant is added to the sign extended exponent and is driven to the exponent input register. At the same time the original exponent is checked to see if it is all ones or zeros. The format converters take one cycle to execute and cause a stall of the next instruction issuing but do not cause stalls in prior instructions.

3.3. Conversion from Hex Internal to Binary Architected

The conversion back to binary from hex internal is performed in the rounder / format convert unit and involves the following formulation:

$$\begin{aligned}
0.hfrac * 16^{Chi-8192} &\Rightarrow 1.X_m * 2^{Cba-b'} \\
0.hfrac &= 0.hfrac * 2^r * 2^{-r} \\
0.hfrac &= 1.X_m * 2^{-r} \\
0.hfrac * 16^{Chi-8192} &
\end{aligned}$$

$$\begin{aligned}
&= 1.X_m * 2^{-r} * 2^{4*Chi-4*8192} \\
&= 1.X_m * 2^{-r} * 2^{4*Chi-2^{15}} * 2^{(2^N-1)} * 2^{-(2^N-1)} \\
&= 1.X_m * 2^{-r} * 2^{4*Chi-2^{15}+2^N-1 \;-\; (2^N-1)} \\
&= 1.X_m * 2^{4*Chi-2^{15}+2^N-(r+1) \;-\; b'} \\
Cba &= 4 * Chi - 2^{15} + 2^N - (r+1)
\end{aligned}$$

$$\begin{aligned}
Let\;\; RSE(A) &= A - 2^{15} + 2^N \\
Cba &= RSE(4 * Chi - (r+1))
\end{aligned}$$

RSE function is basically a reverse sign extension function. Exponents within the range of representable numbers are guaranteed to have C_0 followed by $\overline{C_0}$ for the bit locations from the most significant characteristic position of the internal format to the bit weighted by 2^N. Thus the reverse sign extension can easily be accomplished by just wiring the most significant bit of the hex internal characteristic to the most significant bit of the binary architected format. Or the subtractor width can be minimized to the target characteristic length and the most significant bit can be inverted since it will be one of $\overline{C_0}$ bits. Here is the mapping between formats for different binary normalizations within the hex format, or stated another way, various values of r.

$$\begin{aligned}
(0.0001||X_m) * 16^{Chi-8192} &\Rightarrow 1.X_m * 2^{RSE(4*Chi-5)-b'} \\
(0.001||X_m) * 16^{Chi-8192} &\Rightarrow 1.X_m * 2^{RSE(4*Chi-4)-b'} \\
(0.01||X_m) * 16^{Chi-8192} &\Rightarrow 1.X_m * 2^{RSE(4*Chi-3)-b'} \\
(0.1||X_m) * 16^{Chi-8192} &\Rightarrow 1.X_m * 2^{RSE(4*Chi-2)-b'}
\end{aligned}$$

3.4. Rounder

The rounder is also new to the FPU. It performs an increment of the mantissa in parallel with determining whether the result should be truncated or incremented. The rounding direction is dependent on the least significant digit, the guard digit, the aligner's sticky bit, the normalizer's sticky bit, and a sticky bit within the least significant hex digit determined in the rounder cycle. After the incremented or truncated result is selected, it is binary aligned and the most significant one is chopped off. During this mantissa rounding, the exponent is converted as shown in the previous section. The exponent is also calculated with an increment of one which is selected if the mantissa rounding results in an exponent increment.

3.5. Other BFP hardware

The major hardware change is to add format converters on the input and output and a rounder so that binary data can be treated as though it is in hex internal format. There are two other functions that are needed: sticky bit detection, and special number handling.

Sticky bit detection is needed so that it can be determined whether there are any bits that have been shifted out of the intermediate result. The intermediate result is maintained with several guard bits but this is not enough for the IEEE 754 standard. It must be known how the intermediate result differs from the infinitely precise result. The only places in the G5 FPU dataflow where precision can be lost are in the aligner and the normalizer. For both cases the sticky bit is calculated in parallel with the shifting by calculating all the possible sticky bit outcomes and selecting between these outcomes. Shifting is performed to a hex boundary so there are only 15 possible shifts for the aligner (long) and 29 possible shifts (extended) for the normalizer. The sticky bit is maintained in the pipeline to a hex digit boundary until the rounder.

A special number handler is also needed and is part of the input format converts, the control logic, and the rounder. The input format converts detect if a special number is present. Also the dataflow can signal to controls that underflow or overflow has occurred that may result in a zero or infinite result. The control logic decides if a special number should be forced. If one is to be forced, the operation is continued until the last cycle of execution, the rounder cycle, and this is where the special number is created. This is true for infinities and zeros but denormalized inputs are handled in the dataflow as normal numbers; the format converter automatically suppresses the implied one and the exponent is forced to a fixed value. Also, the presence of a NaN input changes the chosen operation, such as a multiply or add, into a load operation. The result when an input operand is a NaN is the NaN itself (except for possibly flipping a bit which changes a signaling NaN to a quiet NaN). The creation of a new NaN (i.e. 0 divided by 0) is also performed in the rounder.

Creation of special numbers is straightforward for zeros and infinities and even NaNs. But creating a denormalized result is difficult due to the late detection. For pipelined instructions the detection is performed partially in the normalizer and has a final resolution in the rounder stage which is the last stage in the pipeline. The intermediate result at this stage is normalized and must be right shifted back to the fixed denormal exponent. This requires feeding the intermediate value back through the pipeline to the normalization stage. If there are other instructions in the pipeline a serialization exception is taken and the instruction is reissued in unoverlapped mode. If there are no other instructions in the pipeline the normalized intermediate result (no prior rounding) is wrapped to the top of the pipeline and right shifted and rounded.

Thus, to support binary floating point on a hex dataflow, two format converters are added below the input registers, a rounder / format convert is added above the output register, stickyness is detected on the aligner and normalizer, and a special number handler is added.

4. Performance

As mentioned earlier, the dataflow is optimized for hex instructions but with compatibility and reasonable performance for binary instructions. Table 3 shows the performance of a few instructions where L indicates la-

Instruction	Type	Format	Execution L	Execution T
Load	H/B	S	2	1
Load	H/B	L	2	1
Add/Subtract	H	S	3	1
Add/Subtract	H	L	3	1
Add/Subtract	H	X	12	12
Add/Subtract	B	S	5	2
Add/Subtract	B	L	5	2
Add/Subtract	B	X	20	20
Multiply	H	S	3	1
Multiply	H	L	3	1
Multiply	H	X	16	16
Multiply	B	S	6	2
Multiply	B	L	6	2
Multiply	B	X	27	27
Multiply/Add	B	S	13	13
Multiply/Add	B	L	18	18

Table 3. Performance of Instructions

tency and T throughput. In general hex short and long data can be pipelined 1 instruction per cycle and binary short and long data can be input 1 instruction every 2 cycles. The limitation for binary is the result of optimizing the wiring of A and B input data busses and placing the input format converters without an input staging register. Instead A and B registers are used two cycles in a row, creating a stall following binary instructions. Most hex short and long instructions have a latency of 3 cycles but binary short and long instructions for addition take 2 additional cycles for input format conversion and rounding / output format conversion. Binary multiply short and long requires 1 more cycle for a total of 6 cycles, since it requires a cycle through the normalizer to do sticky bit detection while hex multiply normally does not use the normalizer. Binary multiply/add has a latency of 13 cycles for short and 18 cycles for long which is longer than the two operations separately. In the S/390 G5 machine it is suggested this instruction be used if only one rounding can be tolerated. To perform a long multiply/add with one rounding requires a multiply long times long to extended followed by an extended add followed by a load rounded from extended to long precision. The fused multiply/add instruction is faster than these three instructions. Future implementations of multiply/add will probably have similar performance to the RS/6000 implementation. The current value of this instruction is rather limited but it provides early support of the instruction on a machine without a multiply/add dataflow.

This machine competes in commercial environments rather than scientific. The performance is limited to 1 binary instruction every 2 cycles but the clock rate is rather high at 500 MHz and the machine comes in a 10 way shared memory multiprocessor configuration. The hex performance can be up to twice as high as the binary performance but in most applications the performance difference is less noticeable. This hardware implementation replaces a software solution which is either not bit for bit compatible or is extremely slow.

5. Conclusion

The S/390 G5 mainframe computer is the first processor to implement both the traditional S/360 hexadecimal floating point format and the IEEE 754 binary floating point standard. We have shown the new S/390 architecture which supports both architectures, and an internal format which supports both. The S/390 floating point architecture has dramatically grown from 54 opcodes to 175 opcodes. The internal format is optimized for hex floating point which is our current market and it supports the growing market of IEEE 754 floating point. Also, the dataflow modifications were detailed including deriving the format conversion equations and discussing other necessary hardware. The simple add-on type design enabled us to implement a huge architecture change and still meet a highly aggressive schedule of a follow-on machine.

Acknowledgments

The authors would like to thank the contributions of Leon Sigal, Robert Averill, Thomas McPherson, Sean Carey, Fanchieh Yee, Barry Winter, Rick Dennis, Dave Webber, Charles Webb, Jeff Li, Mike Mullen, and Kai-Ann Mueller.

References

[1] G. M. Amdahl, G. A. Blaauw, and J. F. P. Brooks. "Architecture of the IBM System/360," *IBM Journal of Research and Development*, 8(2):87–97, April 1964.

[2] "IEEE standard for binary floating-point arithmetic, ANSI/IEEE Std 754-1985," The Institute of Electrical and Electronic Engineers, Inc., New York, Aug. 1985.

[3] "PA-RISC 1.1 Architecture and Instruction Set Reference Manual," available at http://www.hp.com/, February 1994.

[4] "Alpha Architecture Handbook Reference Manual," Order Number EC-QD2KB-TE, available at http://ftp.digital.com/pub/, October 1996.

[5] "Enterprise Systems Architecture/390 Principles of Operation," Order No. SA22-7201-5, available through IBM branch offices, Sept 1998.

[6] Robert Averill et. al. "Deep submicron design techniques for the 500 MHz IBM S/390 G5 custom microprocessor," In *ICCD'98*, Austin, TX, October 1998.

[7] E. M. Schwarz, B. Averill, and L. Sigal. "A radix-8 CMOS S/390 multiplier," In *Proc. of 13th Symp. on Comput. Arith.*, pages 2–9, Asilomar, CA, July 1997.

[8] E. M. Schwarz, L. Sigal, and T. McPherson. "CMOS floating point unit for the S/390 parallel enterpise server G4," *IBM Journal of Research and Development*, 41(4/5):475–488, July/Sept. 1997.

[9] Timothy Slegel et. al. "IBM S/390 G5 microprocessor," In *Hot Chips 10*, Stanford, CA, August 1998.