VARIABLE-LENGTH DATA FORMATS

Brakefield, James C., Technology Incorporated
Houston, Texas 77058
Quinn, Matthew J., NASA-Johnson Space Center
Houston, Texas 77058

ABSTRACT

From Information Theory, we learn that variable word length data formats can be more efficient than their constant word length counterparts. However, in the past, constant word length computers were designed because the hardware complexity demanded by variable word length machines offset any possible savings by completely filling up the memory. With the advent of LSI, many old concepts are no longer valid. The purpose of this paper is to discuss a number of variable length floating point and integer formats and to give the various advantages and disadvantages of their use. Often one knows in advance that a given integer will not exceed a certain magnitude or that a particular floating point number is only accurate to only "n" places of accuracy. Faced with this, it is good engineering to choose variable length floating point and integer formats which require the least amount of hardware or the minimum amount of software or which have some other dominant advantage. The formats discussed have the advantage that length change algorithms are invariant with respect to data types (unsigned, signed, floating point, integers, and complex numbers). Finally one particular computer, the STARAN associative array processor, uses a completely variable fixed point and floating point formats. Its hardware and software are described to show how a variable length data format is a practical goal.

VARIABLE LENGTH DATA FORMATS

This paper describes the different fixed length formats in use throughout the computer industry. Integer formats are covered in one's complement, two's complement, and binary coded decimal. Then floating point numbers are discussed for a number of the popular computers. Finally, the hardware and software of the STARAN computer, one of the few variable word length computers in the industry, is described in some detail. Specific integer and floating point formats are recommended as most suitable for mathematical operations.

VARIABLE LENGTH FLOATING POINT FORMATS

The following develops a number of "theorems" about equivalent number representation and formats. The notation is a variation on that of reference 1.

r is the base, typically 2, 4, 8, 10 or 16

ℓ is the number of bits used in a base r number and includes any sign or exponent.

US(r,ℓ) is an unsigned number to base r using ℓ bits

ISM(r,ℓ) is a sign and magnitude integer

IOC(r,ℓ) is a one's complement integer

ITC(r,ℓ) is a two's complement integer

F(r,ℓ) is a fractional number to base r using ℓ bits

US, SM, OC, and TC will also be used as suffixes

FP(r,ℓ,e,m,a) is a floating point number of ℓ bits with the mantissa normalized via base r. e is the exponent representation and m the mantissa representation. a is U for unnormalized and N for normalized.

$\approx$ is an associative and commutative relation expressing approximate equality of two number representations.

$\lessapprox$ is like $\approx$ except that the two operands need not be equally dense.

For Unsigned Numbers:

1.1 $0 \leq US(2,\ell) \leq 2^{\ell-1}$

1.2 $0 \leq US(10,\ell) \leq 10^{\ell-1}$

Since $2^m = 10^{m/\log_2 10}$

1.3 $US(2,m) \approx US(10,4m/\log_2 10) \approx US(10,1.2m)$

Which shows that base 10 requires 20% more bits to represent the same range of values.

The following will be used later:

2.1 $US(2,\ell) \approxeq 2^* US(2,\ell-\log_2 2) \approxeq 3^* US(2,\ell-\log_2 3) \approxeq 4^* US(2,\ell-\log_2 4)$

2.2 So: $2^{US(2,\ell)} \approxeq {}_4US(2,\ell-1) \approxeq {}_8US(2,\ell-1.585) \approxeq {}_{16}US(2,\ell-2)$

For Integers:

3.1 $-2^{\ell-1} + 1 \leq ISM(2,\ell) \leq 2^{\ell-1}-1$

3.2 $-2^{\ell-1} + 1 \leq IOC(2,\ell) \leq 2^{\ell-1}-1$

3.3 $-2^{\ell-1} \leq ITC(2,\ell) \leq 2^{\ell-1}-1$

3.4 Thus: $US(2,\ell) - 2^{\ell-1} \approx ISM(2,\ell) \approx IOC(2,\ell) \approx ITC(2,\ell)$

3.5 Similarly: $I\ 'x'\ (2,\ell) \approx F\ 'x'\ (2,\ell) * 2^{\ell-1}$

Note that ISM and IOC are symmetrical and thus contain a "-0" code. ITC is not symmetrical, but the "$-2^{\ell-1}$" code is not generally useful as "$-(-2^{\ell-1})$" results in an overflow.

Floating Point:

Most computer floating point is with binary numbers, the only variations being the choice of exponent and mantissa representations and the radix used in normalizing the mantissa.

Examining the different exponent and mantissa representations:

4.1 $FP(r,\ell,US(s,k) - 2^{k-1}, m,a) \approxeq$

$FP(r,\ell,ISM(s,k), m,a) \approxeq$

$FP(r,\ell,IOC(s,k), m,a) \approxeq$

$FP(r,\ell,ITC(s,k), m,a)$

4.2 and

$FP(r,\ell,e,ISM(s,k), a) \approxeq$

$FP(r,\ell,e,IOC(s,k), a) \approxeq$

$FP(r,\ell,e,ITC(s,k), a) \approxeq$

FP(r,ℓ,e + k-1, FSM(s,k), a)≈

FP(r,ℓ,e + k-1, FOC(s,k), a)≈

FP(r,ℓ,e + k-1, FTC(s,k), a)

Which means that one can use either a biased unsigned exponent or a signed integer exponent of any type with little difference in the range of floating point values which can be represented. Also the mantissa can be of any of the integer types. If the mantissa is of any of the fraction types the effect is to offset the exponent by the mantissa length (without sign).

Now for different r using unnormalized FP with specific e and m for illustration, (using 2.2):

5.1 FP(16,ℓ + 1, ISM(2,k-2),ISM(16,ℓ-k+3), U)≈

FP(8,ℓ + .415, ISM(2,k-1.585),ISM(8,ℓ-k+2), U)≈

FP(4,ℓ,ISM(2,k-1),IAM(4,ℓ-k+1), U)≈

FP(2,ℓ,ISM(2,k), ISM(2,ℓ-k), U)

The summary of the above is that base 8 and 16 waste .415 and 1.0 bits respectively compared to base 2. This happens because although the exponent requires fewer bits to represent the same range, the mantissa requires even more bits to keep the same accuracy. Thus convenience in normalization costs in either more bits or less accuracy.

The final point is that for FP(2,ℓ,e,m,N) an additional bit can be saved since the leading digit of the mantissa must be 1 bit and thus need not be stored. Various fractions of a bit can in theory be saved in the other radicies (using 5.1).

6.1 $FP(2,\ell,e,m,U) \approx FP(2,\ell-1,e,m,N) \approx$

$FP(4,\ell,e,m,U) \approx FP(4,\ell-\log_2 4/3,e,m,N) \approx$

$FP(8,\ell+.415,e,m,U) \approx FP(8,\ell-\log_2 8/7+.415,e,m,N) \approx$

$FP(16,\ell+1,e,m,U) \approx FP(16,\ell-\log_2 16/15 + 1,e,m,N)$

Thus $FP(2,\ell-1,e,m,U)$ is optimum from the storage cost/fixed accuracy point of view.

Now we ask the question: What is the best format for variable length quantities? This is in theory a hardware or software engineering question which would have some relation to the "gate costs" of a particular technology. We will submit that truncation is the simplest method of shortening a quantity and the zero extension is the simplest method of lengthening a quantity. The following formats have the property that only truncation on the left is used for shortening and that only zero extension on the left is used for lengthening.

7.1 $US(2,\ell) = m_{\ell-1}\,2^{\ell-1} + m_{\ell-2}\,2^{\ell-2} \ldots + m_1 2^1 + m_0 2^0$

$= m_{\ell-1}, m_{\ell-2}, \ldots m_1, m_0$

7.2 $ISM(2,\ell) = (1-2*s)(m_{\ell-2}2^{\ell-2} + m_{\ell-3}2^{\ell-3} \ldots + m_1 2^1 + m_0 2^0)$

$= m_{\ell-2}, m_{\ell-3}, \ldots m_1, m_0, s$

7.3 $FP(2,4k,ISM(2,k), FSM(2,3k) + 1,N) =$

$(1-2*m_s)(1+m_{-1}2^{-1}+m_{-2}2^{-2}+ \ldots m_{-3k+2}2^{-3k+2}+m_{-3k+1}2^{-3k+1})*2\uparrow$

$(e_{k-2}2^{k-2} + e_{k-3}2^{k-3} + \ldots e_1 2^1 + e_0 2^0)*(1-2*e_s)$

$= m_{-3k+1}, m_{-3k+2}, m_{-3k+3}, e_{k-2} \ldots e_1 m_{-5} m_{-4} m_{-3} e_0 m_{-2} m_{-1} m_s e_s$

Ie, sign and magnitude with sign on the left, floating point with exponent and mantissa interleaved, and with the mantissa bit reversed. And! the hardware need only know the new and old "ℓ" and not whether the quantity is US, ISM, or FP. The best interleaving factor, on the basis of required exponent range appears to be one exponent bit to three mantissa bits.

EXAMPLE OF A VARIABLE WORD LENGTH COMPUTER

The STARAN (Goodyear Aerospace Corporation) is an example of a variable word length computer. This computer can have thirty two array processors with 256 processing elements in each array. At the NASA-Johnson Space Center, the STARAN consists of two arrays allowing 512 operations to be performed simultaneously. Each processing element can perform any one of the basic sixteen Boolean functions of two logic variables. Each array consists of 256 by 256 bits of memory with 256 processing elements. Each processing element accepts the three inputs from the X response store register, the Y response store register, and the M response store register. (Refer to Figure 1). The X and Y registers are used to perform the arithmetic operations. The M register is used to indicate which words of the array participate in the masked operation. When the mask bit is set, the programmed operation between X and Y is performed. When the mask bit is zero, no operation is performed between X and Y. The combination of one X bit, one Y bit, and one M bit trigger the arithmetic unit that outputs a bit to the 256 by 256 bit memory.

Since the computer can perform any one of the sixteen basic Boolean functions, the programmer has great flexibility in what he can do. He can build the overall program from the elementary Boolean building blocks. While this gives him considerable flexibility, it requires longer and more complex programs. To ease the programmer's work, a higher order language, APPLE (Array Processor Programming Language) is available with a considerable number of macros to perform frequently used functions. By combining Boolean operations, the more complex mathematical operations can be performed.

Addition, subtraction, multiplication, and division can be performed to any degree of accuracy using machine language or APPLE instructions. Furthermore, they can be done with reasonable programs. The convenient limit of 256 bits for addition and subtraction and 128 bits for multiplication and division is currently available in the STARAN at this time. To double these lengths of operands is still a reasonable program. This flexibility makes the STARAN an ideal candidate to simulate computer accuracy.

By programming a number of different solutions to problems involved with different word lengths, the computer architect would be free to explore the limitations of his paper design before he builds the prototype. The normal method of analyzing accuracy of a matrix inversion using an n x n Hilbert Matrix, can be done using variable word length numbers to attain the required accuracy. No longer does one need to analyze accuracy budgets to determine if a particular word length gives the desired accuracy. By programming a general problem with a variable word length and variable parameters, actual accuracy of the computer can be determined. In the case of an n x n Hilbert Matrix, the variable n can be varied as well as the word length of the numerical elements of the matrix array. This gives the computer architect a two dimensional picture of the accuracy of his hardware and software system.

VARIABLE WORD LENGTH EXAMPLE

The STARAN computer uses a variable word length format called S.I.F. where S is the sign, I is the integer and F is the fractional element. For example, i.8.7 indicates one sign bit, 8 integer bits, and 7 fractional bits.

Another example, i.7.0. refers to one sign bit and seven integer bits, whereas 1.0.23 refers to one sign bit and 23 fractional bits. In these examples, the length was never over 24 bits. However, there is no limit on the word length. Basically 256 bits on addition and subtraction, and 128 bits on multiplication and division is a practical word length constraint. However, the length can be extended by clever manipulation of the STARAN to include even greater word length if it should be necessary.

In summary, a number of different formats has been presented. A method of obtaining these formats is available. By programming the STARAN computer for variable word length, essentially any format is available to the computer architect.

References:

Sterbenz, Pat H., Floating Point Computation, p. 12, 1974 Prentice-Hall.

IBM 360 FP(16,32,US(2,7)-64,FSM(16,25))

Data General FP(16,32,US(2,7)-64,FSM(16,25))

SEL FP(16,32,US(2,7)-64,FSM(16,25))

Xerox Sigma 5 FP(16,32,US(2,7)-64,FSM(16,25))

Bendix G20 FP(8,29,ISM(2,7),ISM(8,22))

Burroughs 5500 FP(8,47,ISM(2,7),ISM(8,40))

IBM 1620 FP(10,5n+10,ISM(10,10),FSM(10,5n))

Burroughs 2500 FP(10,4n+16,ISM(10,12),ISM(10,4n+4))

PDP 11 FP(2,32,US(2,8)-128,FSM(2,24)+1)

HP 3000 FP(2,32,US(2,9)-256,FSM(2,23)+1)

PDP 10 FPTC(2,36,US(2,8)-128,FUS(2,27))

GE 635 & Honeywell 6000 FP(2,36,ITC(2,8),2*FTC(2,28))

Univac 1100's FPOC(2,36,US(2,8)-128,US(2,27))

IBM 7090 FP(2,36,US(2,8)-128,FSM(2,28))

Burroughs 8500 FP(2,48,ISM(2,12),ISM(2,36))

CDC 1604 FPOC(2,48,US(2,11)-1024,FUS(2,36))

Harris-Datacraft FP(2,47,ITC(2,8),FTC(2,39))

Philco 213 FP(2,48,ITC(2,12),FTC(2,36))

CDC 6600 FPOC(2,60,US(2,11)-1024,US(2,48))

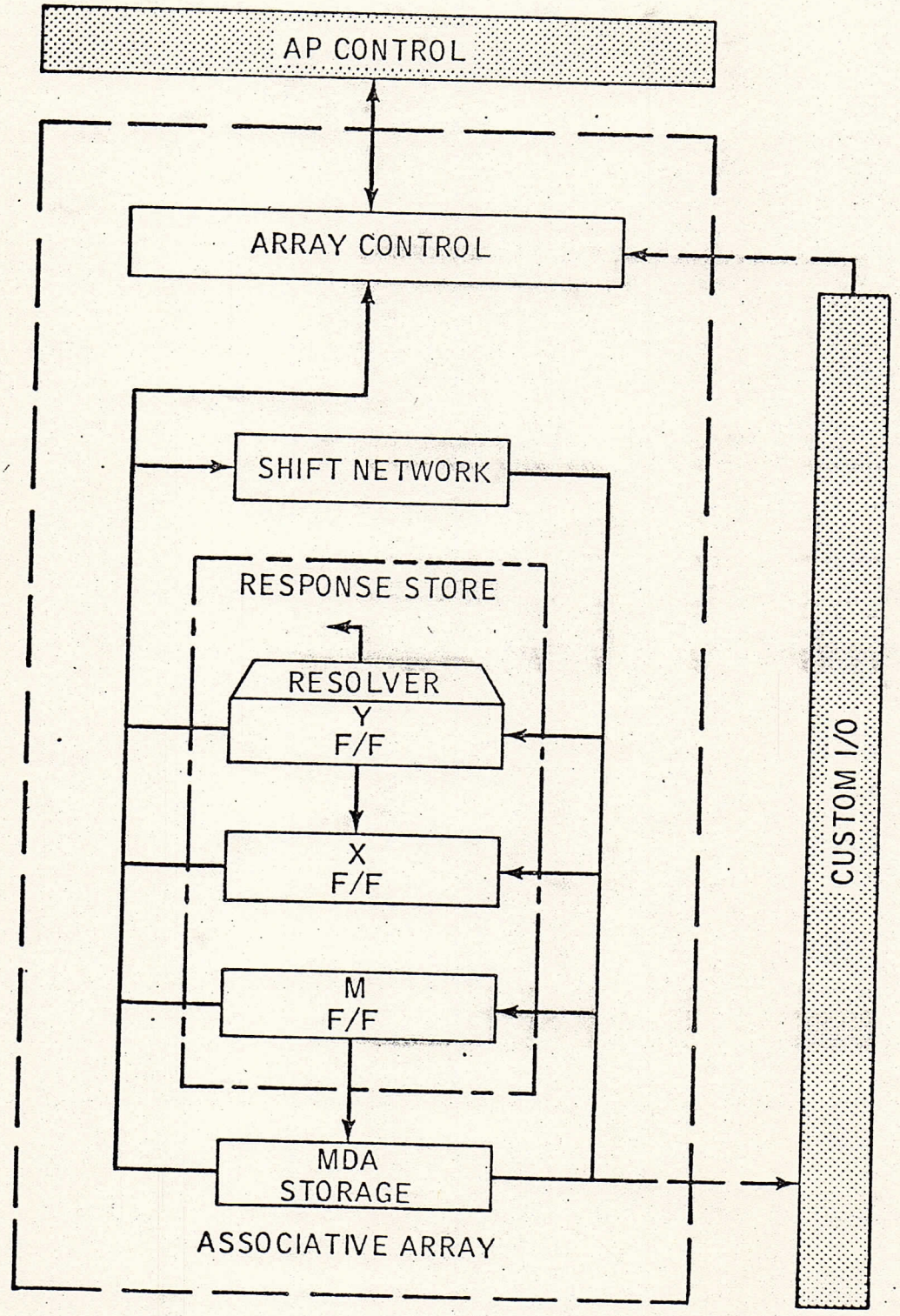
CRAY 1 FP(2,64,US(2,15)-16384,FSM(2,49))

CDC STAR FP(2,64,ITC(2,16),ITC(2,48))

IBM 7030 FP(2,60,ISM(2,11),FSM(2,49))

ILLIAC IV FP(2,64,US(@,14)-8192,ISM(2,50))

Optimal FP(2,4n,ISM(2,n),FSM(2,3n)+1.)


AP CONTROL
ARRAY CONTROL
SHIFT NETWORK
RESPONSE STORE
RESOLVER
Y
F/F
X
F/F
M
F/F
MDA
STORAGE
ASSOCIATIVE ARRAY
CUSTOM I/O