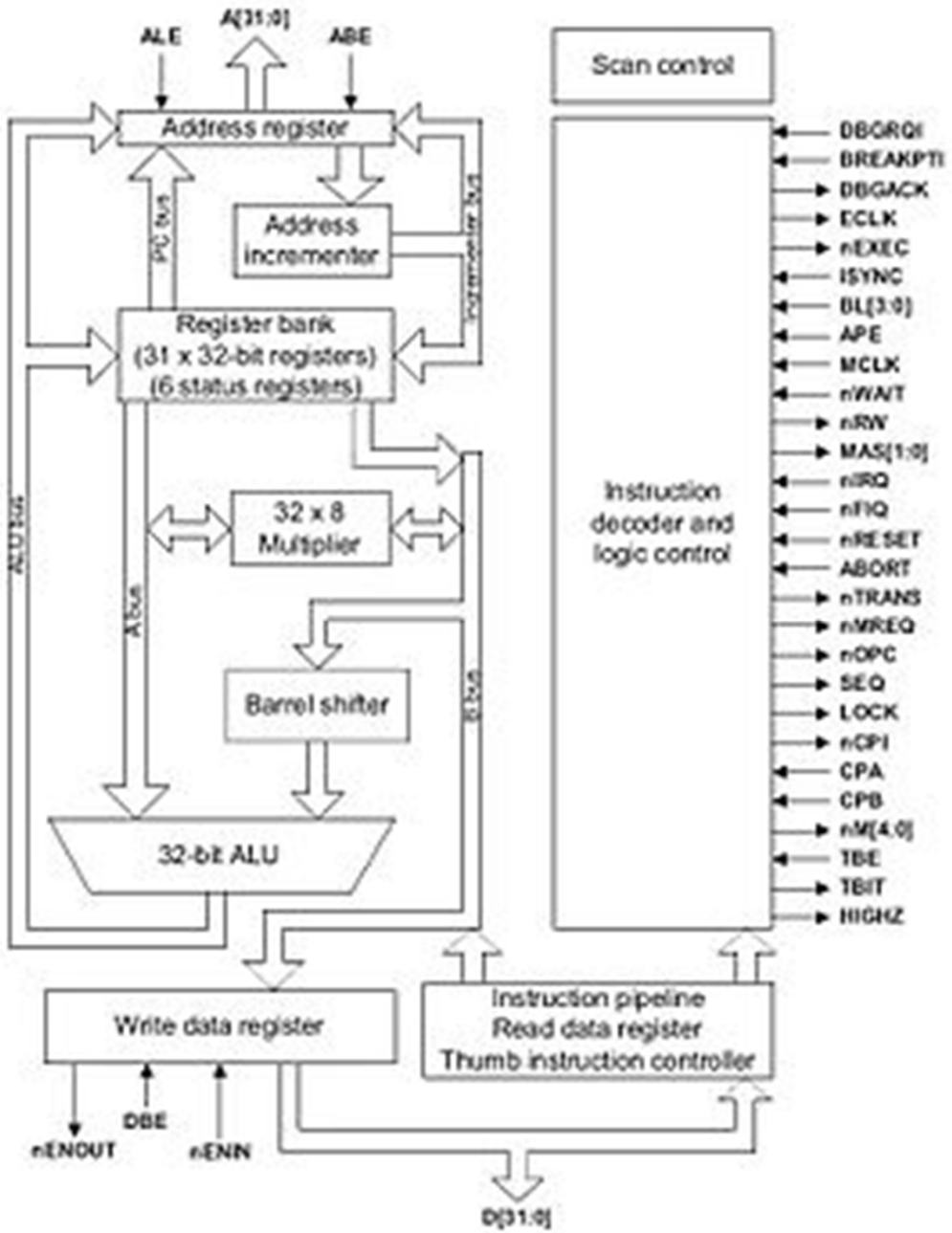
Instruction Sets and Data Representation

- Instruction set defines the interface between software modules and underlying hardware.

“Nothing is ever achieved without enthusiasm.”
-- Ralph Waldo Emerson

ARM7TDMI : Top Level Architecture



Announcement on April 7,1964 by IBM

The term *Computer Architecture* was first defined in the paper by Amdahl, Blaauw and Brooks of International Business Machines (IBM) Corporation announcing IBM System/360 computer family on April 7, 1964.

On that day IBM Corporation introduced, in the words of IBM spokesman, *"the most important product announcement that this corporation has made in its history"*.

- *Instruction set*
- *Instruction format*
- *Operation codes*
- *Addressing modes*
- *All registers and memory locations that may be directly manipulated or tested by a machine language program*
- *Formats for data representation*

Complex Instruction set computers

- Provide instructions that may take many cycles and perform complex tasks such as string searching.
- CISC was primarily motivated by Idea was to reduce the "semantic gap" between the machine language of the processor and the high-level languages in which people were programming
- CISC computers are based on a complex instruction set in which instructions are executed by microcode.

Reduced Instruction Set Computers

- Provide instructions executed in pipelined processors ideally in one cycle.
- Early RISC designs outperformed CISC designs of that period.
- Performance gap between RISC and CISC vanishes if we can use RISC instructions to execute CISC instructions using better compilers

CISC Beginnings: IBM

- The IBM 360 system, created in 1964: probably the first modern processor system so-called Complex Instruction Set Computer (CISC).
- Initiated the idea of computer architecture in computer science and adopted micro-coded control.
- Micro-coded control facilitated the use of complex instruction sets and provided flexibility
- Microcode allows developers to change hardware designs and still maintain backward compatibility with instructions for earlier computers by changing only the microcode,

CISC- Advantages in 1964

- Execute fewer instructions and thus would have better performance
- Hardware was extremely expensive, thus fewer memory occupy was strongly preferred.
- Compiler technology at that time was in its infancy, same as advanced programming language, people always used assembly language at that time,
- CISC made people no need consider the influence of compiler to CPU performance.

CISC- Drawbacks

- Microcode slows microprocessor performance because of the number of operations that must be performed to execute each CISC instruction.

- A CISC instruction set typically includes many instructions with different sizes and execution cycles, which makes CISC instructions harder to pipeline.

- From the 60's CISC microprocessors became prevalent, each successive processor having more and more complicated hardware and more and more complex instruction sets. This trend still continues today from Intel 80486, Pentium MMX to Pentium III and ..

RISC Beginnings:" fresh *look at existing ideas"*

- Middle of 70's, people began to doubt the design philosophy behind CISC more and more
- Complex instructions sets, decoding and execution of such instructions were complicated and time-consuming.
- Expensive overhead brought by them slowed down the execution of those more frequently used simple instructions.

“You just need people with the desire to better themselves.”
Adam Cooper and Bill Collage

RISC Research : Initial Success :IBM

- In 70's, John Cocke at IBM's T.J Watson Research
- Center provided the fundamental concepts of RISC, the idea came from the IBM 801 minicomputer built in 1971 which is used as a fast controller in a very large telephone switching system.
- This chip contained many traits a later RISC chip should have: few instructions, fix sized instructions in a fixed format, execution on a single cycle of a processor and a Load / Store architecture.
- These ideas were further refined and articulated by a group at University Of California Berkeley led by David Patterson, who coined the term "RISC".
- They realized that RISC promised higher performance, less cost and faster design time.

RISC Research : Work at Berkeley

- These ideas were further refined and articulated by a group at University Of California Berkeley led by David Patterson, who coined the term "RISC".

- They realized that RISC promised higher performance, less cost and faster design time.

Features of RISC Architecture

Table 1. Features of RISC Architecture

Feature	Characteristic
Load / Store Architecture	All of the operations are Register to Register. In this way Operation is decoupled from the access to memory
Carefully selected sub-set of instructions	Control is implemented in hardware. There is no microcoding in RISC. Also this set of instructions is not necessarily small*
Simple Addressing Modes	Only the most frequently used addressing modes are used. Also it is important that they can fit into the existing pipeline.
Fixed size and fixed fields instructions	This is necessary to be able to decode instruction and access operands in one cycle. Though there are architectures using two sizes for the instruction format (IBM PC-RT)
Delayed Branch Instruction (known also as Branch and Execute)	The most important performance improvement through instruction architecture. (no longer true in new designs)
One Instruction Per Cycle execution rate, CPI = 1.0	Possible only through the use of pipelining
Optimizing Compiler	Close coupling between the architecture and the compiler. Compiler *"knows"* about the pipeline.
Harvard Architecture	Separation of Instruction and Data Cache resulting in increased memory bandwidth.

Most Misunderstood Part : RISC instruction set is smaller than CISC instruction set

- This number of RISC instructions can grow until the complexity of the control logic begin to impose an increase in the clock period.
- IBM PC-RT Instruction architecture 118 instructions. (RISC type)
- IBM RS/6000 (PowerPC) 184 instructions. (RISC type)
- IBM System/360 contains 143 instructions Not RISC
- IBM System/370 contains 208. : Not RISC

Comparison Between RISC and CISC

Table 3.0 Summary of 80386 and MIPS R2000 architectures

	MIPS R2000	Intel 80386
Date announced	1986	1985
Instruction size(bits)	32	Variable
Address space(size, model)	32 bits, flat	32 bits, segmented with paging support
Data alignment	Aligned	No
Data addressing modes	2	11
Protection	Page	Segmented Scheme
Integer registers (number, model, size)	31 GPR*32 bits	8 GPR*32 bits, 6 segment registers*16 bits, 2 other * 16 bits
Separate floating-point registers	16*32 or 16*64 bits	8*80 bits
Floating-point format	IEEE 754 single, double	IEEE 754 single, double, extended

MIPS: Microprocessor without Interlocked Pipeline Stages

Benchmark selection : Integer Benchmarks

- *test1*: Test1 is a program from Nullstone Corp. which is part of a compiler optimization analysis program.

- *Stanford*: This is a suite of benchmarks that are relatively short, both in program size and execution time. It includes 7 small programs: bubble sort, towers of Hanoi, the 8 queens problem, quick sort, integer matrix multiply, Perm and Sieve. The unusually small size of the codes over emphasizes the benefit of small caches.

- *poly*: It is a very common and typical real-world integer program by a beginning programmer,

Benchmark selection: Floating-point Benchmarks

:

• *whetstone*: Whetstone consists of a mix of floating-point and integer arithmetic, function calls, array indexing, conditional jumps and transcendental functions. Whetstone has been carefully arranged to defeat many simple compiler optimizations. It makes little use of memory and benefits little from the addition of cache memory.

• *var*: Corresponding to *poly*, it is a very common real-world FP program by a beginning programmer.

• *lre*: This is a real-world computation intensive program used in EE for research of Partial likelihood for Channel Equalization.

Frequency of Instructions and Instruction classes

Table 3.1a 80386 instruction mix for three integer benchmarks

Instruction	Stanford	Poly	Test1	Average
Mov(Load)	28.87%	44.47%	41.91%	38.42%
Move(Store)	11.51%	10.31%	11.46%	11.09%
Add	9.36%	15.99%	3.84%	9.73%
Sub	2.23%	0.73%	0.04%	1%
Mul	2.49%	3.24%		1.91%
Div	0.01%			
Cmp	9.83%	5.32%	15.26%	10.14%
Mov(Reg-Reg)	2.11%	0.83%	0.08%	1.01%
Mov(Imm)	1.27%	0.76%	0.08%	0.70%
Push, pop	11.93%	5.31%	0.37%	5.87%
Cond branch	9.57%	5.17%	15.29%	10.01%
Uncond branch	5.62%	6.17%	3.84%	5.21%
Call	2.71%	0.71%	0.04%	1.15%
Ret	0.11%	0.42%	0.04%	0.19%
Shift	0.09%	0.04%		0.04%
And, or	1.68%	0.04%		0.57%
Other(Xor, not…)	0.60%	0.49%	7.53%	2.87%
FP load				
FP store				
FP add				
FP sub				
FP mul				
FP div				
FP cmp				
FP Others				

Frequency of Instructions and Instruction classes

Figure 3.1b 80386 instruction mix for three floating point benchmarks

Instruction	Whetstone	Lre	Var	Average
Mov(Load)	26.28%	38.38%	20.92%	28.53%
Move(Store)	11.51%	7.43%	9.96%	9.63%
Add	4.26%	4.87%	8.98%	6.04%
Sub	2.8%	0.10%	4.69%	2.53%
Mul	1.44%	4.89%	2.34%	2.89%
Div				
Cmp	3.26%	4.30%	4.27%	3.94%
Mov(Reg-Reg)	2.99%	0.19%	0.04%	1.07%
Mov(Imm)	2.99%			1%
Push, pop	17.68%	1.47%	4.36%	7.83%
Cond branch	3.25%	4.57%	4.29%	4.04%
Uncond branch	4.18%	5.26%	4.59%	4.68%
Call	2.08%	0.76%	4.43%	2.42%
Ret	1.49%	0.18%	0.04%	0.57%
Shift				
And, or				
Other(Xor, not…)	1.49%	0.28%	0.03%	0.6%
FP load	2.65%	5.33%	8.98%	5.65%
FP store	4.13%	5.65%	8.35%	6.04%
FP add	3.60%	4.39%	5.40%	4.46%
FP sub	0.40%	0.45%	1.83%	0.89%
FP mul	2.23%	10.28%	3.61%	5.37%
FP div	1.11%	0.51%	2.68%	1.43%
FP cmp		0.28%	0.03%	0.1%
FP Others	0.09%	0.28%	0.03%	0.13%

Frequency of Instructions and Instruction classes

Figure 3.2a MIPS2000 instruction mix for three integer benchmarks

Instruction	Stanford	Poly	Test1	Average
Load	34.21%	29.24%	26.27%	29.91%
Store	16.02%	13.79%	0.09%	9.97%
Add	18.85%	29.58%	13.14%	20.52%
Sub	1.76%	0.16%		0.64%
Mul	0.21%	2.62%		0.94%
Div				
Compare	6.15%	4.22%	8.69%	6.35%
Load imm	2.49%	0.79%	21.52%	8.27%
Cond. Branch	7.49%	8.69%	17.29%	11.16%
Uncond branch	1.73%	0.22%	4.28%	2.08%
Call	0.85%	0.68%	0.04%	0.52%
Return, jmp ind	0.88%	0.73%	0.04%	0.55%
Shift	8.66%	9.18%	8.56%	8.8%
And	0.14%	0.09%		0.08%
Or	0.56%		0.08%	0.21%
Other(xor,not)		0.01%		
Load FP				
Store FP				
Add FP				
sub FP				
Mul FP				
div FP				
Compare FP				
Mov reg-reg FP				
Other FP				

Frequency of Instructions and Instruction classes

Figure 3.2b MIPS R2000 instruction mix for the three floating point benchmarks

Instruction	Whetstone	Lre	Var	Average
Load	25.91%	8.31%	37.25%	23.83%
Store	5.78%	2.98%	4.86%	4.54%
Add	11.62%	28.19%	8.5%	16.1%
Sub	0.85%		2.52%	1.12%
Mul	1.19%			0.4%
Div				
Compare	2.16%	1.52%	3.03%	2.24%
Load imm	3.95%	19.45%	2.99%	8.79%
Cond. Branch	4.53%	3.08%	3.68%	3.76%
Uncond branch	0.42%	0.03%		0.15%
Call	1.73%	0.26%	1.03%	1.01%
Return, jmp ind	1.73%	0.26%	1.03%	1.01%
Shift	3.35%	5.57	3.61%	4.18%
And		0.01%		
Or		4.89%	0.96%	1.95%
Other (xor, not)			0.96%	0.32%
Load FP	17.52%	12.99%	6.74%	12.42%
Store FP	11.89%	4.54%	2.66%	6.36%
Add FP	2.92%	1.52%	2.43%	2.29%
Sub FP	0.42%	0.2%	0.79%	0.47%
Mul FP	1.86%	3.6%	1.64%	2.37%
div FP	0.93%	0.18%	1.21%	0.77%
Compare FP		0.1%	0.02%	0.03%
Mov reg-reg FP	0.88%	1.55%	10.88%	4.44%
Other FP	0.34%	0.77%	3.18%	1.43%

Top 10 Instructions for 80386

Table 3.1c Top 10 integer instructions for 80386

Rank	Instruction	% total
1	Move	51.22%
2	Compare	10.14%
3	Cond branch	10.01%
4	Add	9.73%
5	Uncond branch	5.21%
6	Push	2.94%
7	Pop	2.94%
8	Mul	1.91%
9	Call	1.15%
10	Sub	1%
Total		**96.25%**

Conclusions

• IC in 80386 is less than MIPS R2000.

• CISC has richer Instruction set, but in our case, the programs just use around 15% of all the instructions

• CISC has better code density, 80386's average instruction length is less than MIPS R2000's

• 80386 has richer addressing modes, but in both integer and floating-point case, 3 addressing modes used for about 90%'s addressing, some addressing modes even never used

• To 80386, the operand types can be "Memory", while in MIPS R2000, only load/store will access memory, all operands are in registers. Cache and memory designer should be aware of this difference.

Who Wins in the Future?

- Difference between RISC and CISC chips is getting smaller. RISC and CISC architectures are becoming more and more alike. Many of today's RISC chips support just as many instructions as yesterday's CISC chips

- Today, both RISC and CISC manufacturers are doing everything to get an edge on the competition.

- Biggest threat for CISC and RISC might not be each other, but a new technology called EPIC. EPIC stands for Explicitly Parallel Instruction Computing.

- EPIC created by Intel is a combination of both CISC and RISC. This will in theory allow the processing of Windows-based as well as UNIX-based applications by the same CPU.

- Intel is working on it under code-name Merced. Microsoft is already developing their Win64 standard for it. Like the name says, Merced will be a 64-bit chip.

A Small Dream

- Will a team of IIITB students develop the first processor core from India using enhanced version of EPIC :Explicitly Parallel Instruction Computing to achieve low power and high throughput?

http://semiaccurate.com/2011/06/20/china-develops-mips-processor-for-smartphones/

China develops MIPS processor for smart phones.
Uses only 240mW with integrated graphics

“A person who never made a mistake never tried anything new.”
--- Albert Einstein

Principles of Fixed Point Arithmetic

In computing arithmetic, fractional quantities can be approximated by using a pair of integers *(n, e)*: the mantissa and the exponent. The pair represents the fraction:

$$n2^{-e}$$

The exponent *e* can be considered as the number of digits you have to move into *n* before placing the binary point.

For example:

01100100	-1	011001000.	200
01100100	0	01100100.	100
01100100	1	0110010.0	50
01100100	2	011001.00	25
01100100	3	01100.100	12.5
01100100	7	0.1100100	0.78125

Table 2-1: Principles of fixed point arithmetic

Fixed and Floating Point Arithmetic

•Floating point number: If e is a variable quantity, held in a register and unknown at compile time, (n, e) is said to be a floating point number.

• Fixed point number : If e is known in advance, at compile time, (n, e) is said to a fixed point number.

• Fixed point numbers can be stored in standard integer variables (or registers) by storing the mantissa.

Change of exponent

The simplest operation to be performed on a fixed point number is to change the exponent. To change the exponent from *p* to *r* (to perform the operation *c* = *a*) the mantissa *k* can be calculated from *n* by a simple shift.

Since:

$$n2^{-p} = n2^{r-p} \times 2^{-r}$$

you have the formula:

```
k = n << (r-p)      if (r>=p)
k = n >> (p-r)      if (p>r)
```

Addition and subtraction

To perform the operation $c = a+b$, first convert a and b to have the same exponent r as c and then add the mantissas. This method is proved by the equation:

$$n2^{-r} + m2^{-r} = (n+m)2^{-r}$$

Subtraction is similar.

Multiplication

The product $c = ab$ can be performed using a single integer multiplication. From the equation:

$$ab = n2^{-p} \times m2^{-q} = (nm)2^{-(p+q)}$$

it follows that the product $n*m$ is the mantissa of the answer with exponent $p+q$. To convert the answer to have exponent r, perform shifts as described above.

For example, if $p+q \geq r$:

```
k = (n*m) >> (p+q-r)
```

Division

Division, $c = a/b$, can also be performed using a single integer division. The equation is:

$$\frac{a}{b} = \frac{n2^{-p}}{m2^{-q}} = \left(\frac{n}{m}\right)2^{q-p} = \left(\frac{n}{m}\right)2^{(r+q-p)}2^{-r}$$

In order not to lose precision, the multiplication by $2^{(r+q-p)}$ must be performed before the division by m.

For example, assuming that $r+q \geq p$, perform the calculation:

```
k = (n<<(r+q-p))/m
```

Square root

The equation for square root is:

$$\sqrt{a} = \sqrt{n2^{-p}} = \sqrt{n2^{(2r-p)}} \times 2^{-r}$$

In other words, to perform $c = \sqrt{a}$, set `k=isqr(n<<(2r-p))` where `isqr` is an integer square root function.

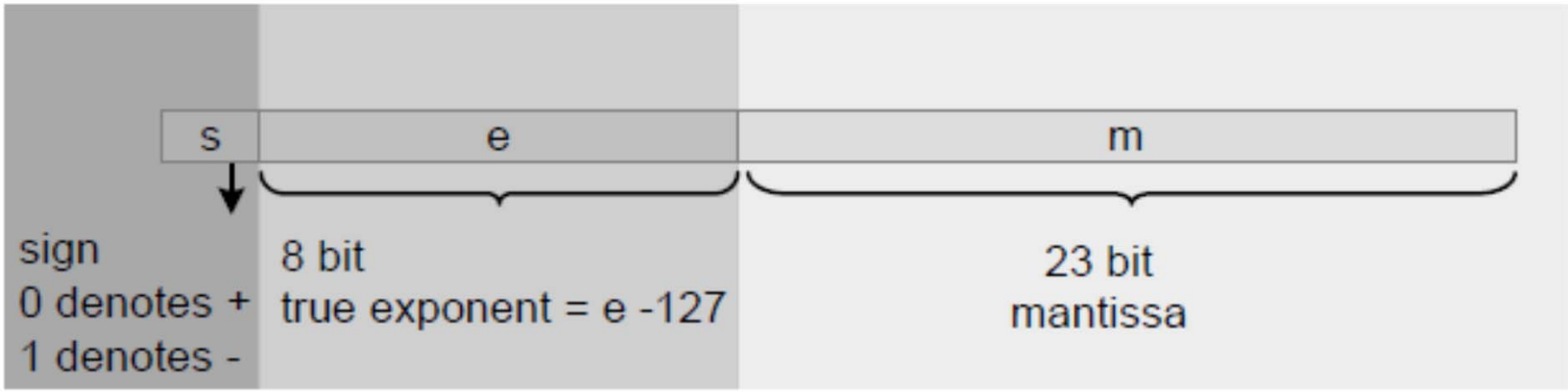
Floating Point Representation

- Floating point arithmetic is appropriate for high precision applications
- Applications that deals with number with wider dynamic range
- A floating point number is represented as

$$x = (-1)^s \times 1 \times m \times 2^{e-b}$$

  - s represents sign of the number
  - m is a fraction number >1 and < 2
  - e is a biased exponent, always positive
    - The bias **b** is subtracted to get the actual exponent

Floating Point Representation : IEEE 754 Format



"Because of the standardization which IEEE-754 (1985) provided, it became possible to write algorithms using floating point arithmetic which could be executed on a variety of platforms and which would produce identical results. It became possible to prove statements about the behavior of floating point programs."

Example of Floating Point Representation

32-bit IEEE Floating point number is

0_10000010_11010000_00000000_0000000

Sign bit	Exponent		Mantissa
0	10000010		11010000_00000000_0000000
$(-1)^0$	$\times\ 2^{(130-127)}$	$\times$	$(1\ .\ 1\ 1\ 0\ 1)_2$
1	$\times\ 2^{(3)}$	$\times$	$(1 + 0.5 + .25 + 0 + 0.0625)_{10}$
(+1)	$\times\ 2^{(3)}$	$\times$	$(1.8125)_{10}$

Decimal floating-point arithmetic as specified in IEEE 754-2008 standard.

Speed: As opposed to computers, users prefer the use of the decimal notation. On certain applications decimal to binary conversions and binary to decimal conversions is so great that decimal operations require 50%-90% of processor time.

Accuracy: With floating-point numbers in binary format, accuracy problems prevail. For instance, the decimal term 0.1 has no finite binary representation: **Dec**: 0.1 = **Bin**: 0.0001100110011....

Uniformity: Today's computation with floating-point numbers might yield diverse results in different processors.

Decimal floating-point arithmetic: IEEE 754-2008 standard.

“In 2000, the IEEE chartered a new committee to examine the IEEE-754 standard with the goals of including decimal floating point arithmetic, incorporating good existing practice, providing for reproducible results, and clarifying the standard, while not invalidating conforming implementations of the IEEE-754(1985) standard”

Decimal floating-point arithmetic as specified in IEEE 754-2008 standard.

S (Sign)	G (Combination field)	T (Trailing Significand field)

The IEEE 754-2008 defines DFP number as : $(-1)^s \times (10)^q \times c$, where: S is the sign bit, q is the exponent, $c = (d_{p-1} d_{p-2} \cdots d_0)$ is the significand where $d_i \in 0,1,2,3,4,5,6,7,8,9$, and p is the precision.

S is the sign bit

G is a combination field that contains the exponent, the most significant digit of the significand, and the encoding classification.

The rest of the significand is stored in the Trailing Significand Field, T, using either the Densely Packed Decimal (DPD) encoding or the Binary Integer Decimal (BID) encoding, where the total number of significand digits corresponds to the precision, p. The DPD encoding represents every three consecutive decimal digits in the decimal significand using 10 bits,.

IEEE 754-2008 standard: Parameters

S (Sign)	G (Combination field)	T (Trailing Significand field)

Parameter	decimal32	decimal64	decimal128
Total storage width (bits)	32	64	128
Combination Field (w+5) (bits)	11	13	17
Trailing significand Field (t) (bits)	20	50	110
Total Significand Digits (p)	7	16	34
Exponent Bias	101	398	6176
Exponent Width (bits)	8	10	14

Binary Floating Point Representation

Represent -28.53 in a 32 bit register with IEEE 754 Floating point format

Find s, e and m such that

$$-28.53 \cong (-1)^s m 2^{e-b}$$

s = 1, m must be between 1 and 2. b = 127

28.53/4 = 1.7844444.. e –b =4, e = 131.

32 bit register bits

110000100110…..

Error in representation

Decimal Floating Point Representation

Represent -28.53 in a 32 bit register with IEEE 754-2008 Floating point format

Find s, e and m such that

$$-28.53 \cong (-1)^s \, m 10^{e-b}$$

s = 1, m represents densely packed decimal digits

28.53 x 100 = 2853 Pack 3 digits with 10 bits. One more digit will require 4 bits

32 bit register bits

Error in representation = 0

Operations with Numbers represented on Processors

- Fixed point operations require much less overhead
  - Cost of the processors is low
- Binary Floating point operations require more book keeping
  -- Cost of the processors is high

- Decimal floating point operations are simple to manage.
  -- Cost of processing ?

References

1. The ARM Architecture Leonid Ryzhyk leonidr@cse.unsw.edu.au June 5, 2006 ARM_Pipelining.pdf

2. **Application Note 33** Fixed Point Arithmetic on the ARM, Document Number: ARM DAI 0033A, ARM_fixedpoint_appsnote.pdf

4. **ARM7TDMI Instruction set** ARM DDI 0084D arm_instructionset.pdf

References

1. Comparison between CISC and RISC Yi Gao, Shilang Tang, Zhongli Ding {ygao1, stang2, zding1}@cs.umbc.edu

2. **Reduced Instruction Set Computers** Prof. Vojin G. Oklobdzija Berkeley, CA 94708