SH4 RISC Microprocessor for Multimedia

Fumio Arakawa, Osamu Nishii, Kunio Uchiyama, Norio Nakagawa
Hitachi, Ltd.

Outline

3DCG: Three Dimension Computer Graphics

SH4 Overview

- Hitachi's SuperH Series Family

- For Consumer Multimedia Systems

Home Video Game, Handheld PC

- Excellent Performance with Consumer Price

300 VAX MIPS

- Excellent 3DCG-Performance with Consumer Price

5.0 M Polygons/sec*

- IEEE 754 Standard Floating-point Architecture

Double-precision with Hardware Emulations

* measured with an original simple geometry benchmark

SH4 Specifications

Technology	**0.25 μm CMOS, 5 Layer Metals**
Voltage	**1.8 V (I/O: 3.3 V)**
Frequency	**167 MHz (internal) / 83,55, etc. MHz (I/O)**
Performance	**300 MIPS (Dhrystone), 1.17 GFLOPS (peak)**
Cache	**8/16 KB (Inst./Data) Direct-mapped**
TLB	**4/64-entry (Inst./Unified) Fully-associative**
Interfaces	**SRAM, DRAM, SDRAM, burst ROM, PCMCIA**
Peripherals	**DMAC, SCI, RTC, Timer**

Pipeline Stages

- Simple Five-stage Pipelines
- Two-way Superscalar

Integer	**Inst. Fetch**	**Inst. Dec. Reg. Read**	**Exec.**	**---**	**Write Back**
Floating Point		**Inst. Dec. Reg. Read**	**1st Exec.**	**2nd Exec.**	**3rd Exec. Write Back**
Load / Store		**Inst. Dec. Reg. Read**	**Addr. Gen.**	**Memory Access**	**Write Back**
Branch		**Inst. Dec. Addr. Gen.**	**Target Inst. Fetch**		

Superscalar Issue Combinations

	INT	FP	LS	BR	BO	NS
Integer (INT)	X	O	O	O	O	X
Floating Point (FP)	O	X	O	O	O	X
Load / Store (LS)	O	O	X	O	O	X
Branch (BR)	O	O	O	X	O	X
Both INT & LS (BO)	O	O	O	O	O	X
Not Superscalar (NS)	X	X	X	X	X	X

INT: Add, Subtract, Shift, etc.
FP: Floating-point Add, Subtract, Multiply, Divide, etc.
LS: Load/Store/Transfer from/to Integer/Floating-point Register, etc.
BR: Branch Always/Conditionally, etc.
BO: Move between Integer Registers, Integer Compare, etc.
NS: Load to Control Register, etc.

Floating-point Arch. Enhancement

- Two Sets of 16 Single Precision Registers
    - The extra set fits 4 by 4 matrix storage
- Length-4 Vector Instructions
    - Inner Product
    - Transform Vector
- Register Pair Load/Store/Transfer Instructions
    - Enough bandwidth for vector operations
- Double Precision Format Mode

Floating-point Instructions

- Common
    - FADD (add)
    - FSUB (subtract)
    - FMUL (multiply)
    - FDIV (divide)
    - FSQRT (Square Root)
    - FCMP (Compare)
    - FNEG (Negate)
    - FABS (Absolute Value)
    - FLOAT (Convert Integer to float)
    - FTRC (Convert float to Integer)
    - FMOV (Move from/to Register)
- Single Precision Mode Only
    - FMAC (multiply-Accumulate)
    - FIPR (Inner Product)
    - FTRV (Transform Vector)
- Double Precision Mode Only
    - FCNVDS (Convert Double to Single)
    - FCNVSD (Convert Single to Double)

Inner Product Instruction

- 16 Registers = 4 (Length-4) Vector Registers

fv0	= (fr0 ,fr1 ,fr2 ,fr3)
fv4	= (fr4 ,fr5 ,fr6 ,fr7)
fv8	= (fr8 ,fr9 ,fr10,fr11)
fv12	= (fr12,fr13,fr14,fr15)

- Operation

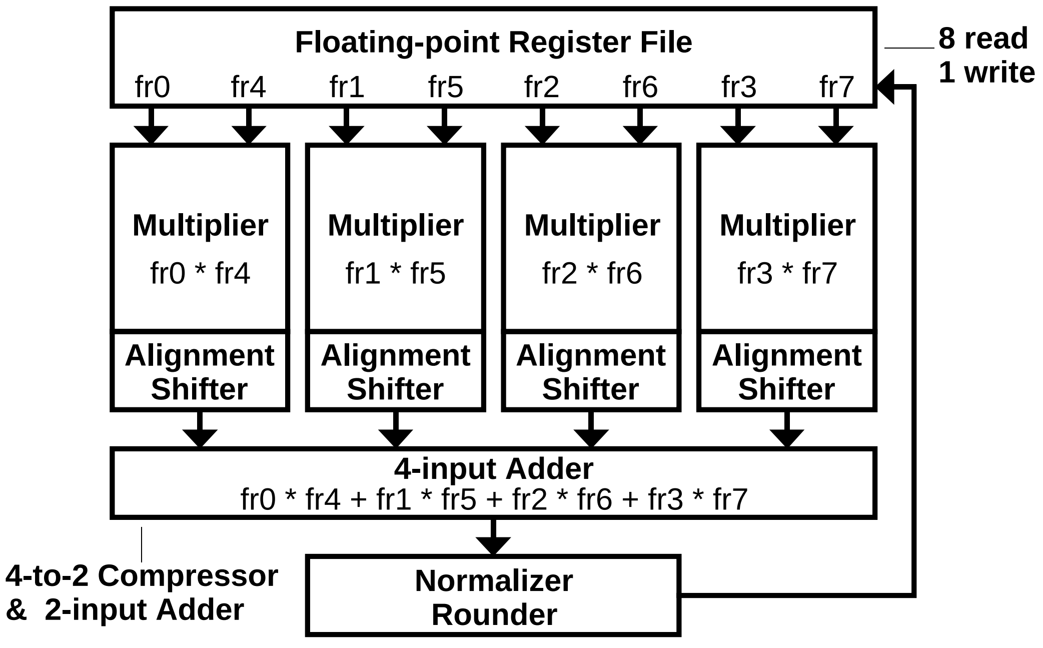
  frn' = (fvm,fvn) m,n: 0,4,8,12 n'= n+3

- 1 cycle Pitch
- 4 cycle Latency
- Vector Normalization, Intensity Calculation, Surface Judgment

HITACHI

Inner Product Hardware (Mantissa)

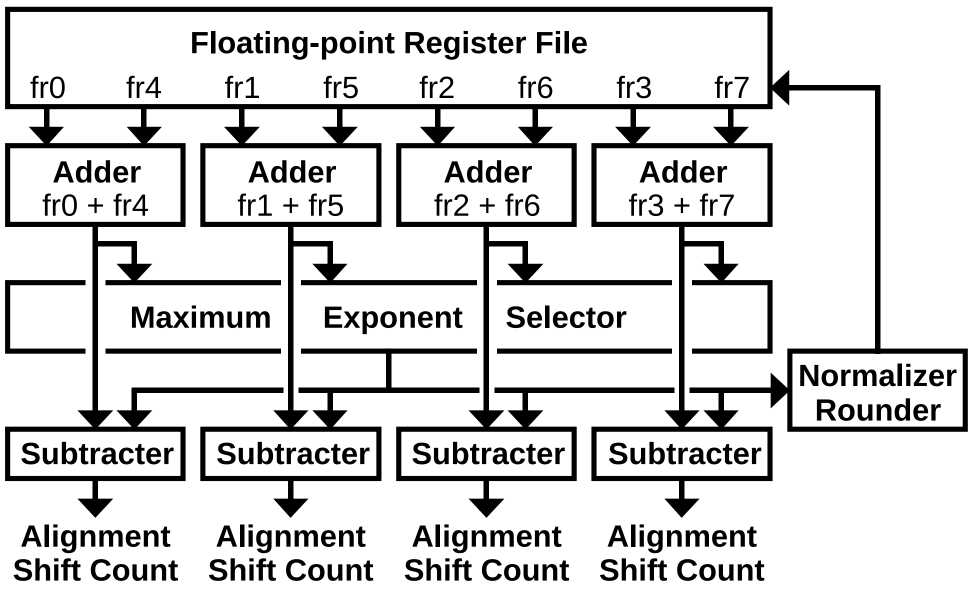
- Calculating fr7 = (fv0,fv4)



HITACHI

Inner Product Hardware (Exponent)

- Calculating fr7 = (fv0,fv4)



Inner Product Accuracy

- Inner Product Inst. is an Approximate Inst.
  - No Accurate Intermediate Value
    (the width is too wide to implement)
  - More Accurate than the Worst Order
    Multiply and Add Inst. Combinations
- Maximum Error:
  (Maximum Product x 2^{-25}) + (Result x 2^{-23})
  - If source operands are rounded values,
    this is enough accuracy.
- For example: 4 Products are 2^{26}, -2^{26}, 1, 0.
  Accurate Result = 1. Inner Product Inst. Result = 0.

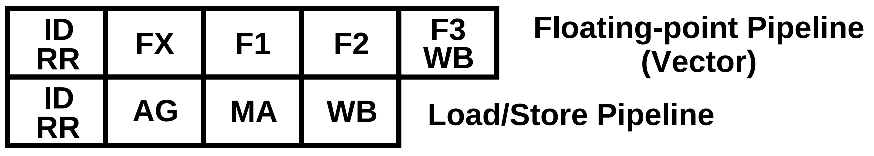
Inner Product v.s. SIMD Multiply-Add

	Inner Product	4 Multiply-Add
Peak Performance	**1**	**1** [1)]
Latency	**4**	**12** [2) 3)]
Register Port (Read/Write)	**8/1**	**12/4**
Normalizer & Rounder	**1**	**4**
Floating-point Hardware	**1**	**2** [4)]

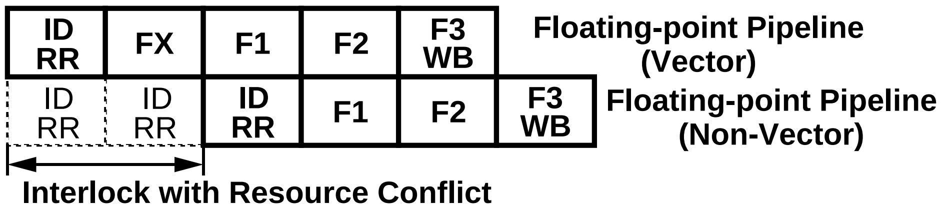
1) Inner Product Inst. and 4 Multiply-Add achieve the same peak performance, which is one inner product per cycle.
2) SH4 takes 3 cycles for Multiply-Add. 4 x 3 = 12.
3) Eight more cycles must be filled with independent insts. for the peak performance with SIMD architecture.
4) Twice more hardware is necessary for SIMD architecture.

Elastic Pipeline

- Pipeline stages become 4 cycles for vector Inst.
- In-Order Issue, Out-of-Order Completion



- Only Floating-point Non-Vector Arithmetic Inst. right after Vector Inst. is interlocked.


ID RR
FX
F1
F2
F3 WB
Floating-point Pipeline (Vector)
ID RR
ID RR
ID RR
F1
F2
F3 WB
Floating-point Pipeline (Non-Vector)
Interlock with Resource Conflict

ID: Instruction Decode, RR: Register Read, FX,F1,F2,F3: Floating-point Execution, AG: Address Generation, MA: Memory Access, WB: Register Write Back

Transform Vector Instruction

- Extra 16 Registers = 4 by 4 Matrix

$$\text{matrix} = \begin{pmatrix} xf0 & xf4 & xf8 & xf12 \\ xf1 & xf5 & xf9 & xf13 \\ xf2 & xf6 & xf10 & xf14 \\ xf3 & xf7 & xf11 & xf15 \end{pmatrix}$$

xf: e<u>x</u>tra <u>f</u>loating-point register

- Operation

  fvn = matrix • fvn n: 0,4,8,12

- 4 cycle Pitch, 7 cycle Latency
- Coordinate Transformation,
  Coordinate Transformation Matrix Generation
- No Work Registers

Why Transform Vector Instruction ?

- Transform Vector Operation = 4 Inner Product Insts. ?

  NO !!

- Modification for Transform Vector:

  frn' = (xvm,fvn)

  m: 0,1,2,3, n: 0,4 n'= n+m+8

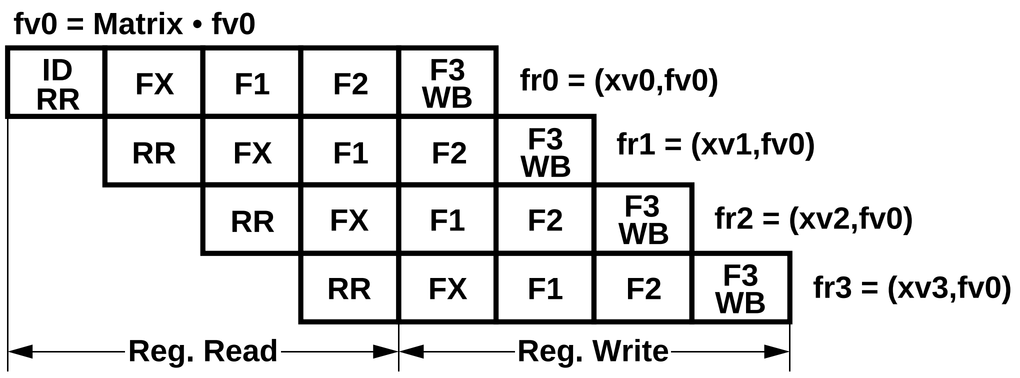
xv0 = (xf0, xf4, xf8 ,xf12)
xv1 = (xf1, xf5, xf9 ,xf13)
xv2 = (xf2, xf6, xf10,xf14)
xv3 = (xf3, xf7, xf11,xf15)

"fv8 = matrix • fv0" is divided into 4 Inner Products:

fr8 = (xv0,fv0)
fr9 = (xv1,fv0)
fr10 = (xv2,fv0)
fr11 = (xv3,fv0)

  - 4 More Work Registers
  - Complicated and More Operands
  - No Generality (Just for Transform Vector)
- Transform Vector Inst. is Better.

Transform Vector Implementation



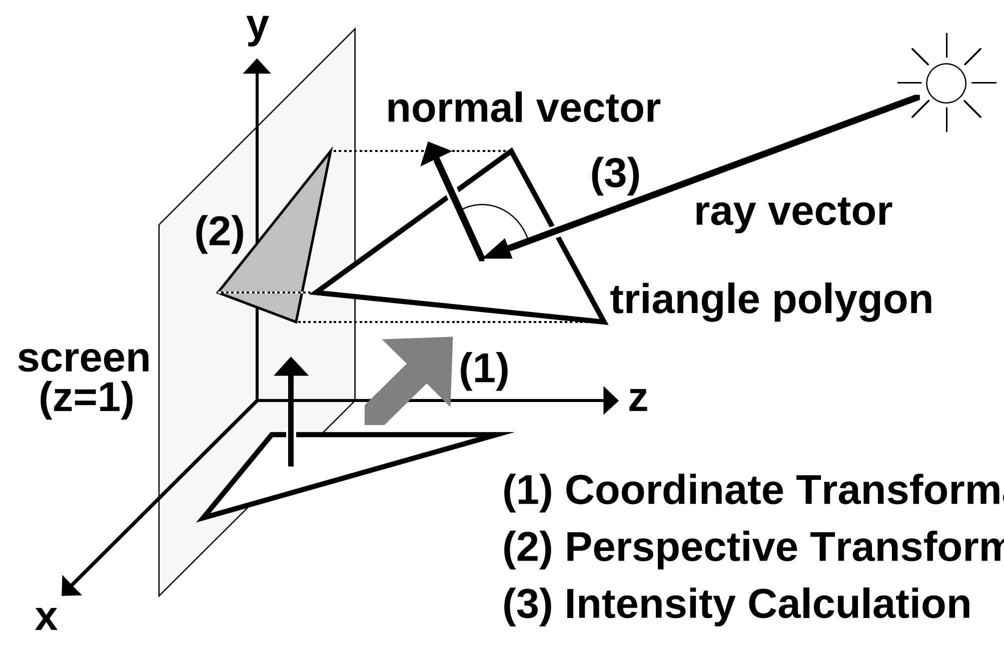
- All reg. reads complete before first reg. write.
No work regs. are necessary.

ID: Instruction Decode, RR: Register Read, WB: Register Write Back
FX,F1,F2,F3: Floating-point Execution

Pair Load/Store/Transfer Mode

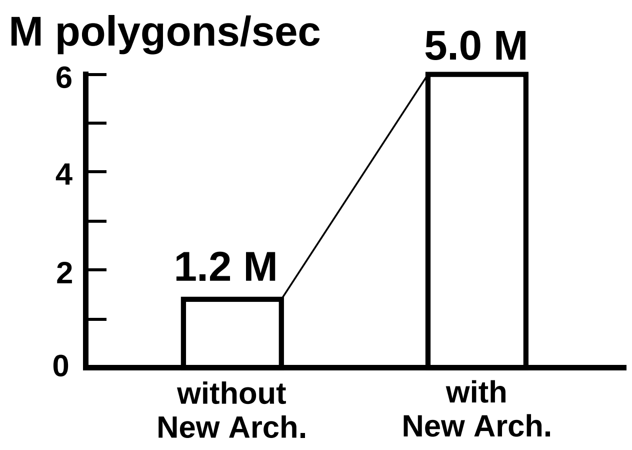
- **Normal Mode:**
  - **4-bit Reg. Field represents 16 Regs. of one set.**
  - **Set specifier must be changed for another set access.**
- **Pair Mode:**
  - **4-bit Reg. Field represents 16 Pair Regs. of all sets.**
  - **All Regs. can be accessed.**
- **Transform Vector Throughput: 1 vector / 4 cycles**
- **Load/Store Throughput: 2 vectors (4 pairs) / 4 cycles**
  - **Enough for Storing Previous Result Vector and Loading Next Vector during Transform Vector**

Simple 3DCG Geometry Benchmark



(1) Coordinate Transformation
(2) Perspective Transformation
(3) Intensity Calculation

3DCG Geometry Performance

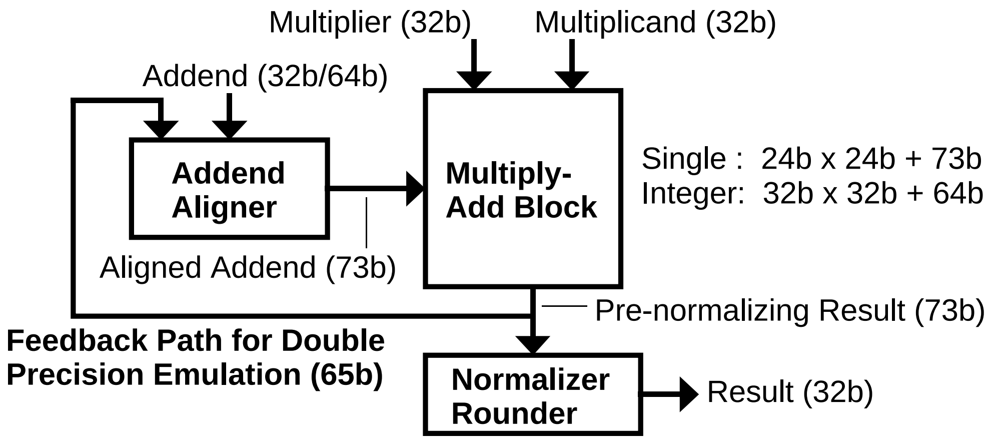


Double Precision Support

- Floating-point Libraries for WindowsCE
    - Double Precision
    - ANSI/IEEE 754 Standard
- Emulation with Single Precision Hardware
    - Best cost-performance way
    - Peak performance is 27.8 MFLOPS.
    - Software emulation is 20 times slower.
    - Double precision hardware is 6 times faster but 2.5 times more.
- New Double Precision Mode
    - Single's code becomes double's code.

Double Precision Hardware

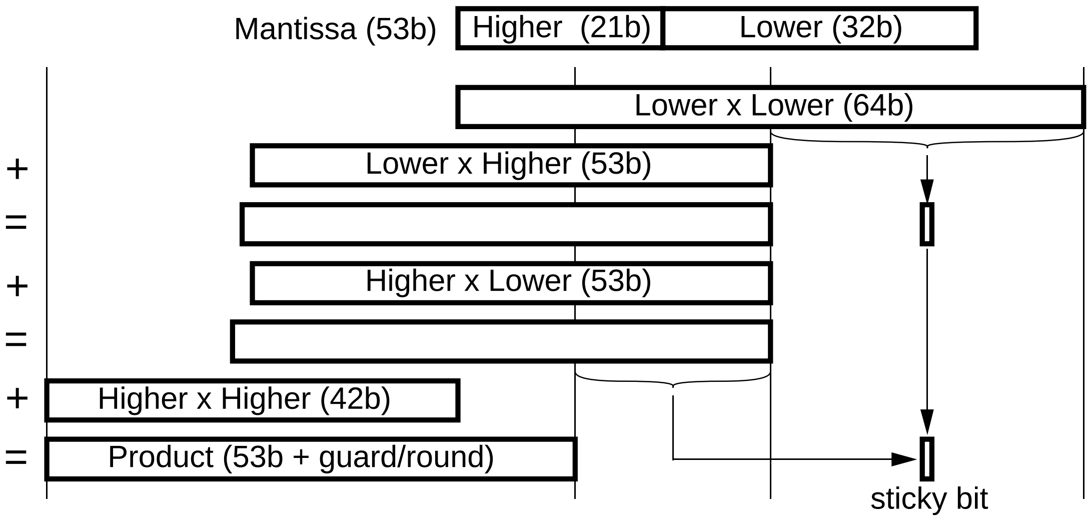
- Mantissa Part
    - Add/Subtract/Multiply/Convert: Add Feedback Path



    - Divide/Square Root: Extend from 24 to 53 bits
- Exponent Part: Extend from 8 to 11 bits

Double Precision Multiply

- Four multiply-adds generate product
- Sticky bit is generated from partial products
    - required width: 106b --> 55b



Conclusions

- Excellent Performance with Consumer Price
    - 300 VAX MIPS
- Excellent 3DCG-Performance with Consumer Price
    - New Inner Product & Vector Transformation Insts.
    - 5.0 M Polygons/sec
    - 1.17 GFLOPS (peak with the new insts.)
- IEEE 754 Standard Floating-point Architecture
    - Double-precision with Hardware Emulations
    - 27.8 MFLOPS (peak)