IBM RISC System/6000

Floating-Point Unit

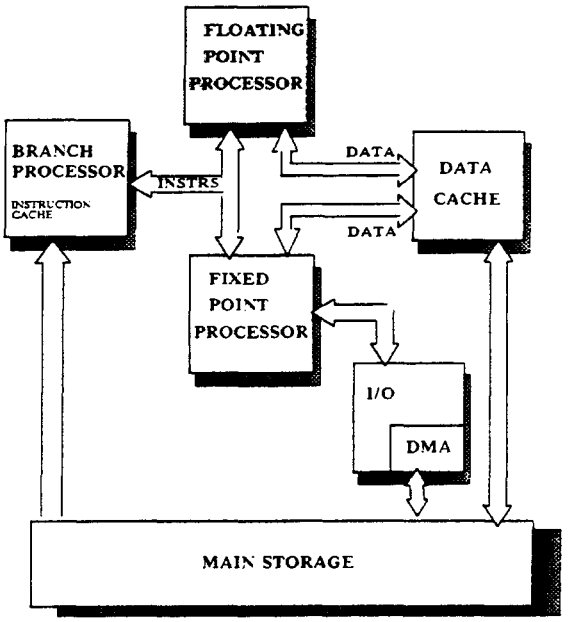
Troy N. Hicks, Richard E. Fry, Oscar R. Mitchell

International Business Machines Corporation
Advanced Workstation Division
Austin, Texas

Contents

- Design Goals
- Implementation
- Performance

Floating-point unit



3

Design Goals

- IEEE Standard 754 Conformance
- Short Pipeline
- High Execution Rate
- Tightly Couped Execution

4

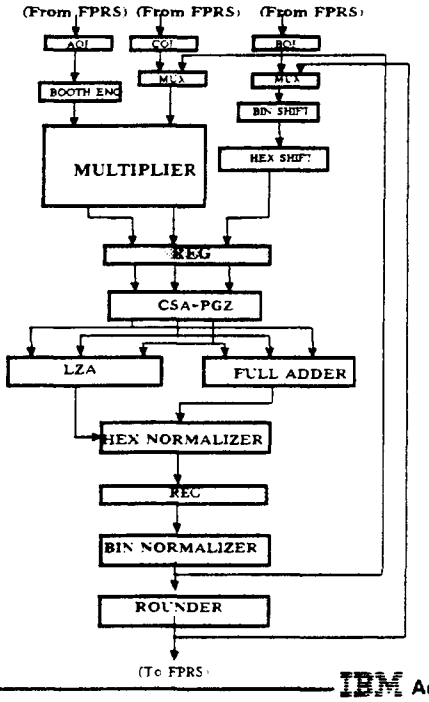
IEEE Standard 754 Conformance

- Four Rounding Modes (+inf, –inf, zero, nearest)
- Single (32 bits) and Double (64 bits) Precision
- All Exceptions Supported (with software assist)

Short Pipeline: Multiply–Add Fused Dataflow

- Dot Product Dataflow
  S = AxC + B
- Reduced Latency
  53–bit x 53–bit Multiply Performed in single Machine Cycle
  106–bit partial products + 54–bit operand in single Machine Cycle
- Leading–Zero Anticipator Predictes Number of Leading Zeros
  In Parallel with Addition
- Only One Round per Mul–Add Operation
  Full 106–bit Partial Products used in Add
- Reduces Internal Busing

Short Pipeline: Multiply–Add Fused Dataflow



Arithmetic Functions Supported in Hardware

fa	add
fs	subtract
fm	multiply
fma	multiply–add
fms	multiply–subtract
fnma	negative multiply–add
fnms	negative multiply–subtract
fabs	absolute value
fneg	negate
fnabs	negative absolute value
frsp	round to single precision
fmr	move
fd	divide

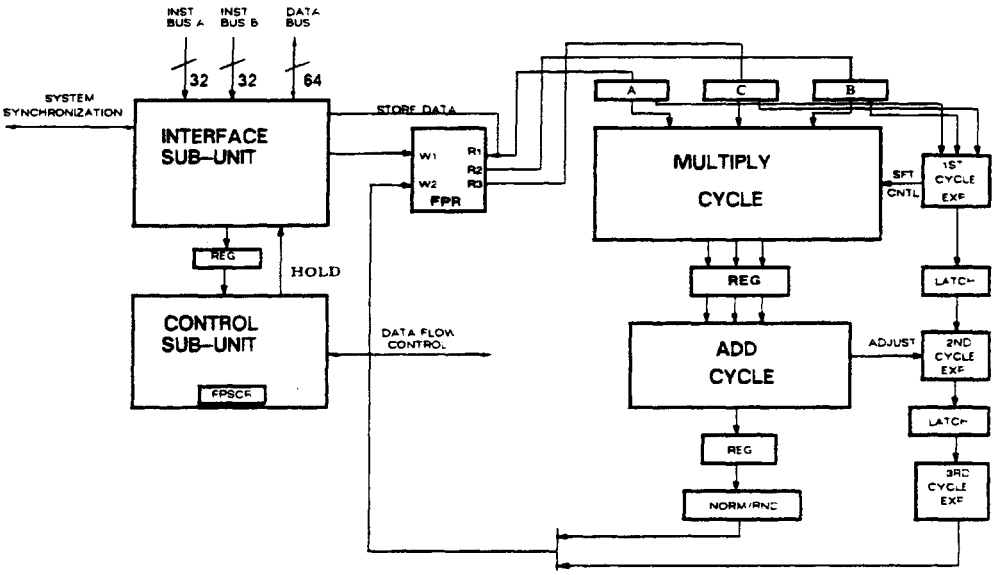
High Execution Rate

- A Double Precision Arithmetic Operation can start every cycle
- Only 2 Cycles of latency
- Addition of first instruction performed in parallel with Multiply of second
- Data Bypasses Cut Delay for back-to-back Dependencies to 1 cycle
- Concurrent Execution of Floating-Point Arithmetic and Load Operations

Tightly Coupled Execution

- FPU and FXU execute instructions concurrently (the FPU is NOT a co-processor)
- FPU and FXU Share Instruction Buses
- FPU and FXU Synchronize on all Interruptable Operations
- Decoupled FPU and FXU Execution Units: Avoid Interlocks
  - Register Renaming
  - Store Data Queue

Floating-point Logical Partitions



//

Register Renaming

OBJECTIVE: Avoid Load/Arithmetic target conflicts in order to eliminate FXPT and DCU hold-offs.

EXAMPLE:

FD 0 1 2

FA 4 0 3

LFD 0

Load can not complete until divide has finished (17-22 cycles).

W/ REMAP:

FD 0 1 2

FA 4 0 3

LFD 0 → LFD 32

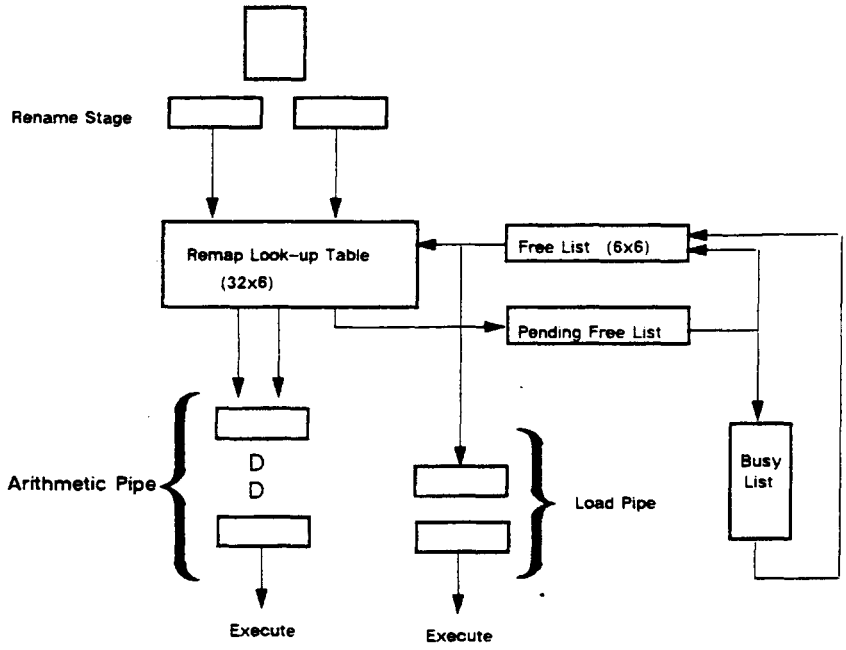
From this point on Architected FPR 0 is physical FPR 32.

NOW THE LOAD DATA MAY ARRIVE AT ANYTIME.

Register Renaming Implementation

- 40 Physical Floating-Point Registers
  - 32 Architected (programmers view)
    - 2 Dedicated for divide routine
    - 6 for Register Renaming
- Table (array) maintains architected-to-physical map
- Table is modified apon completion of Load instruction
- All instruction reference table to "look-up" physical registers used

Register Renaming Implementation



Store Data Queue

- Decouples FPU pipeline from FXU pipeline in executing Store ops
- Store Data is placed in a four deep queue
- Data is moved to DCU (under FXU control) when FPU data bus is not busy
- Execution of subsequent instructions continue

Performance

- 60 MFLOPS peak double precision Linpack at 30MHz (model 530)

Matrix Performance Comparisons:

		time (sec)	ratio
Sparcstation 4/330	(25MHz)	1140.0	(7.8)
DECstation 3100	(16MHz)	1160.0	(7.9)
Convex C-240	(25MHz)	140.0	(1.0)
FPS-500	(33MHz)	170.0	(1.2)
RISC System / 6000	(25MHz)	147.0	—

Summary

- Concurrent execution of fixed point and floating–point operations
- Concurrent execution of floating–point arithmetic and load operations
- Register renaming allows floating loads to execute ahead of arithmetics
- Data store queue allows floating loads to execute ahead of stores
- Multiply–Add instruction provides enhanced precision
- Short pipelined Dataflow performs 60MFLOPS peak at 30MHz