A RISC-V Instruction Set Processor-Micro-architecture Design and Analysis

Aneesh Raveendran, Vinayak Baramu Patil, David Selvakumar, Vivian Desalphine
Centre for Development of Advanced Computing (C-DAC)
Bangalore, India
raneesh@cdac.n, vinayakp@cdac.in, david@cdac.in, viviand@cdac.in

***Abstract*— Micro-architecture design and analysis of a RISC-V instruction set processor has been articulated in this paper. Instruction Set Architectures (ISAs) for processors from Intel, AMD, Intel, MIPS etc. is protected through IP Rights and Infringements. Few ISAs do exist as open-source viz. Open RISC, SPARC, RISC-V etc. RISC-V ISA has been evolved from the efforts at University of California, Berkeley and has been open sourced as BSD license. This paper details the micro-architecture design and analysis of a 5-stage pipelined RISC-V ISA compatible processor and effects of instruction set on the pipeline / micro-architecture design. The design have been analyzed in terms of instructions encoding, functionality of instructions, instruction types, decoder logic complexity, data hazard detection, register file organization and access, functioning of pipeline, effect of branch instructions, control flow, data memory access, operating modes and execution unit hardware resources. The processor has been micro-architected, simulated using Blue-spec System Verilog, synthesized and analyzed on FPGA platform and 65nm and 130nm technology nodes for ASIC. The synthesis results are compared and analyzed with similar efforts on RISC-V ISA based processor core.**

Keywords— Processor Design, Processor Micro-architecture, Out-Of-Order Processor, RISC-V Instruction Set, RISC Processor, IEEE 754-2008 FPU Standard, Floating Point Co-processor.

I. INTRODUCTION

Currently, the IC technology has advanced phenomenally to allow unprecedented implementations of computer systems on a single chip. Custom System on Chip (SoCs), where processors core(s) and cache are small part of the chip become ubiquitous; it is rare today to find an electronic product at any scale that does not include a processor core. Traditionally, instruction set architectures (ISAs) has been proprietary for commercial reasons and there is no known technical reason to release the ISAs as open source. Most of the ISAs from Intel, ARM, AMD, etc. are proprietary, guarded by intellectual property. Open source instruction set architecture like RISC-V [1] and Open RISC [2] based processors have got momentum in custom SoCs designs. The base user level RISC-V ISA from University of California, Berkeley had been released during '11 and the super-user level ISA have been released in '14.

Micro architecture implementation of Reduced Instruction Set Processor (RISC) is more complex and maintaining the Clock per Instruction (CPI) close to 1 is always challenging. The deviation of CPI from ideal value is due to several reasons such as branch miss-prediction, memory latency, pipeline hazards etc. Proposed paper analyses the processor micro architecture design and effect of ISA on pipeline and micro-architecture design with RISC-V ISA. The pipeline has 5-stages viz. Fetch, Decode, Register Select, Execute and Write Back. Pipeline is organized in such a way that integer and floating point execution units are independent to enable concurrent execution of integer and FP computations.

Section II briefs published literature on the topic, and Section III briefs RISC-V ISA. Section IV details pipeline organization and micro-architecture of our RISC-V Instruction Set Processor and Section V briefs internal micro-architecture of FP co-processor and Section VI analyzes the RISC-V ISA and its issues with pipeline stages design. Section VII narrates the testing, validation of processor core, logic synthesis and its analysis and section VIII concludes.

II. RELATED WORKS

Currently few open source ISAs (Open RISC, SPARC, RISC-V etc.) available for designing Low power RISC processor targeted on ASIC and FPGA. An implementation of LEON SPARC [3] processor at 28 nm has been evaluated with Ultra Low Voltage standard cell and memories. An instruction set extension on Open RISC ISAs [4] for digital signal processing is implemented and integrated to the base instruction set. In [5], an ISA design criteria has been proposed and synthesis of the processor instruction set from high level description of ISA has been reported.

Article [6] [7] [8] [9] analyses ISA features of RISC-V, brings out the technical specification for open licensed ISA and recommendations for an open standard for SoCs. Article [9], evaluates and compares RISC-V with EPIPHANY ISA [10] on the aspects of integer, load/store, branch and floating point instructions. Low-RISC [11] is developing fully open hardware systems based on RISC-V viz. Z-scale (a 32-bit, 3-stage, single issue, in-order pipeline), BOOM (64-bit out-of-order), FabScalar (superscalar architecture) and a vector processor (on-chip DC/DC converters on 28 nm technology node).

Shakthi project [12]proposes a RISC-V based processors as, C-Class (32-bit, 3-8 stage in-order variant aimed at 50-250 MHz),I-class(64-bit, 1-4 core, 5-8 stages, out of order, aimed at 200-1GHz), M-class(quad-threaded, up to 8 cores, up to 2.5 GHz), S-class(64-bit superscalar, multi-threaded at 1.2-3GHz with 2-16 cores), H-class(64-bit in-order, multi-threaded with 32-100 cores), T –class(security oriented 64-bit variants with tagged ISA) for various different applications. For the works [11] and [12] internal architecture details haven't been reported viz. FPU co-processor, pipeline design internals etc.

978-1-5090-0033-3/16/$31.00 ©2016 IEEE

Our architecture is a 32 bit, single core, 5-stage pipelined processor with single issue, out-of-order FPU/integer computation, and in-order commit / retire. Proposed core has been synthesized on FPGA and 65nm and 130nm technology nodes (Low Leakage &Standard Process) for ASIC. The synthesis results have been compared with similar efforts on RISC-V ISA compatible processors.

III. RISC-V INSTRUCTION SET ARCHITECTURE

A. ISA Characteristics

ISA provides user and supervisor level instructions to enable support for software stacks including Operating System. Base RISC-V ISA has fixed length 32-bit instructions that must be naturally aligned on 32-bit (word) boundaries. RISC-V ISA supports both integer and floating point operations. Floating point operations are compatible with IEEE 754-2008 standard [9].RISC-V ISA is defined as base integer ISA, which doesn't have any architecturally visible delay slots for branch operation and supports optional variable length instruction encoding.

B. Register File organization

RISC-V supports both integer (32, 32-bit) and floating point registers (32, 64-bit) for user and supervisor level processor modes. For single precision floating point data types, the lower 32-bit of floating point register is valid. The register 'R0' is hardwired to zero.

C. Instruction Classification

The RISC-V instructions are categorized based on the functionality and op-code (instruction [6:0]).Base instruction set (RV32) is classified into 8 categories.

TABLE I: INSTRUCTION GROUPING

Category	Op code	Instructions
Branch Inst.	11_000_11	BEQ, BNE, BLT, BGE, BLTU, BGEU
Load Instructions	00_000_11 00_001_11	LB, LH, LW, LBU, LHU FLW, FLD
Store Instructions	01_000_11 01_001_11	SB, SH, SW FSW, FSD
Arithmetic and Logic (ALU) Instructions	00_100_11 01_100_11	ADDI, SLTI, SLTIU, XORI, ORI, ANDI, SLLI, SRLI, SRAI, ADD, SUB, SLL, SLT, SLTU, XOR, SRL, SRA, OR, AND, MUL, MULH, MULHSU, MULHU, DIV, DIVU, REM, REMU
Atomic Operations	01_011_11	LR.W, SC.W, AMOSWAP.W, AMOADD.W, AMOXOR.W, AMOAND.W, AMOOR.W, AMOMIN.W, AMOMAX.W AMOMINU.W, AMOMAXU.W
System Instructions	11_100_11	SCALL, SBREAK, RDCYCLE, RDCYCLEH, RDTIME, RDTIMEH, RDINSTRET, RDINSTRETH, FRCSR, FRRM,FRFLAGS, FSCSR, FSRM,FSFLAGS, FSRMI, FSFLSGSI
Special, unconditional jump, Synchronization	01_101_11 00_101_11 11_011_11 00_011_11	LUI, AUIPC, JAL, JALR, FENCE, FENCE.I

Table I depicts the instruction grouping based on the op-code field and functionality. Both single and double precision floating point operations are classified into FPU related instructions. Each category of instructions is subdivided as subgroup based on the function field (bits [14:12]) in the instruction.

IV. PIPELINE ORGANIZATION AND MICRO ARCHITECTURE OF RISC-V INSTRUCTION SET PROCESSOR

RISC-V instruction set processor adopts typical RISC micro-architecture with in-order fetch, out-of order execution and in-order completion. Proposed architecture comprises of 5 pipeline stages viz. Fetch, Decode, Register Select (RS), Execute (EXE) and Write Back (WB). Fig 1 shows the architecture of 5-stage pipelined processor. In fetch stage, instruction is fetched from instruction memory from location indicated by Program Counter (PC). Instruction memory sends instruction to decode stage for instruction information extraction. The extracted information is forwarded to register select stage. From register select stage, the pipeline is split into three concurrent pipeline units, as integer instructions pass through integer execution pipeline, memory instructions pass through the memory pipeline and floating point instructions pass through the FP execution pipeline unit. Write back stage receives the responses from all of these three concurrent execution stages and based on the scheduling of the instruction, WB stage allows commit the instructions in-order.

A. Fetch Stage

Fig 2 shows the micro architecture of fetch unit. It computes the current PC and predicts the next PC value. Fetch unit has a 32 bit register to hold the current PC value, a 32-bit adder to calculate the next PC by adding 4 to the current PC. Next PC is predicted either as PC+4 or by using a sophisticated branch prediction scheme [12] [13].

Branch prediction scheme consists of Branch Target Buffer (BTB) and Branch Predication (BP) unit. BTB unit outputs a Boolean valid signal along with target PC address. BP unit provides Boolean information which contains whether the PC value is present or not. A multiplexer selects either PC+4 or predicted PC as next PC value based on the valid and prediction signals. The PC and next PC are given to the decoder stage for further processing. Branch predication unit is explained in detail in Section VI.

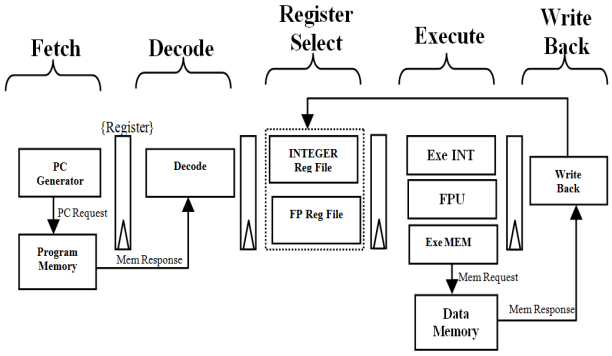

Fig 1: Top level architecture of RISC-V Processor

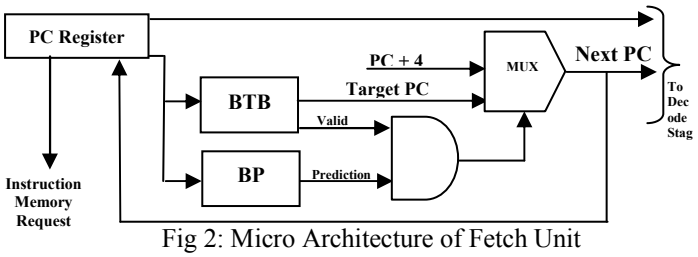

Fig 2: Micro Architecture of Fetch Unit

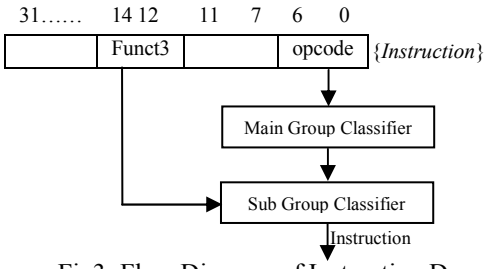

Fig3: Flow Diagram of Instruction Decoder

B. Decode Stage

Instruction decoder receives the instructions from program memory with corresponding PC and predicted PC from fetch stage. Fig 3 depicts the flow diagram of instruction decoder. Based on the lower 7-bit (Op-code) of instruction, instructions are classified as per Table 1. Subgroup information is decoded from 3-bit field (function field). Address of sources and destination registers are of 5-bit field each and their position is fixed irrespective of type of instruction. If an immediate type instruction has occurred, the immediate data is sign-extended to 32-bit for further processing.

Fig 4 shows the micro-architecture of RISC-V instruction decoder. Op-code is given to main group classifier, which gives main group class as output as per Table 1. Main group classifier has seven parallel comparators (7-bit comparator) to decode main group class. Instruction op-code is compared against seven categories (Table 1) of op-codes concurrently. If any of the comparators' output goes high, corresponding main group signal is generated otherwise the instruction being decoded is considered as ILLEGAL. Further, the subgroup classifier determines the exact operation to be performed by the instruction. Subgroup is determined by using both 3-bit function field and main class. If 3-bit function field of instruction matches with 3-bit fields of the decoded main class, sub group classifier generates corresponding subgroup category under the decoded main class, otherwise the instruction falls under ILLEGAL.

Decoder generates control signals for further processing of the instruction. Control signals include, Register Access, Register update, and pipeline information (the pipeline on which the instruction is to be scheduled). Register Access can be integer register file access (INT-ACCESS) or floating point register access (FP-ACCESS) or NO-ACCESS.

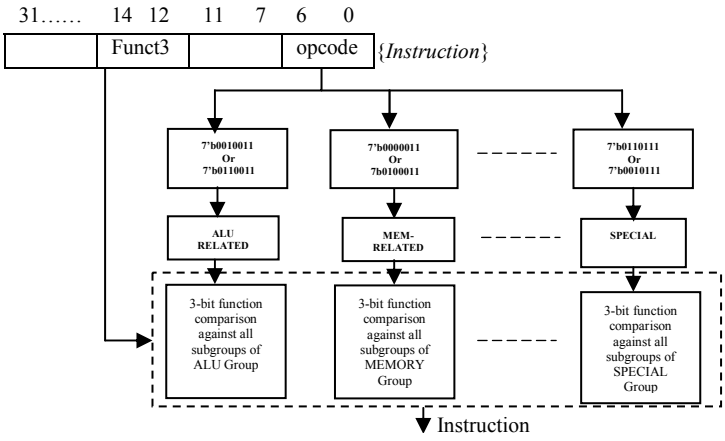

Fig 4: Micro-Architecture of instruction decoder

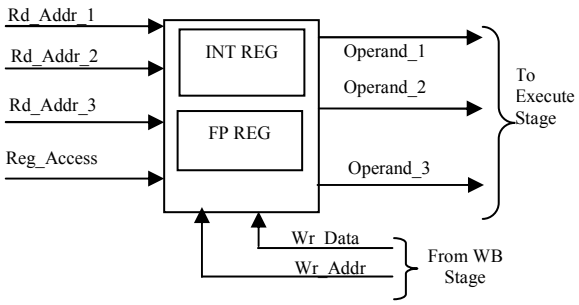

Fig 5: Register File Organization

Register update gives which register file needs to be written back after executing the instruction. Instruction processing is organised through two independent pipelines for integer ALU / Memory Address computation, and FPU. EXPIPE signal indicates the pipeline through which the instruction has to be processed.

Apart from decoding the instruction, decoder detects data dependency (Hazard Detection) between instructions which is explained in section VI. Decoded information is forwarded to register select stage in the form of packet along with PC and predicted PC.

C. Register Select Stage

RS stage accepts decoded information of instructions from decoder stage and selects the operand from either integer or floating point register file. RS stage contains an integer (32-No's of 32-bit wide) and floating point register file (32-No's of 64-bit wide).Register file is implemented as a Random Access Memory (RAM), which has a latency of one clock cycle with three read ports and one write port.

Fig5 shows the register file organization for RISC-V processor. Register file unit accepts three source addresses (Rd_Addr_1, Rd_Addr_2, Rd_Addr_3} and a control signal (Reg_Access) which specifies the access of Register files (Integer Access or Floating point Access). Register file output is 64-bit which holds the data from either integer or floating point register file and has one 64-bit write port for write back.

Based on the pipeline information from decode stage, operands after register select are passed either to integer ALU / memory address computation pipeline or FP execution pipeline. An instruction scheduler FIFO which is part of integer / memory / FPU execute stage is used to schedule the instructions in the various pipelines to commit the instructions in-order.

D. Execute Stage

Execute stage consists of three concurrent units for Integer arithmetic and logic operations etc., operand memory address computation and Floating point computation. Integer execution performs arithmetic (Addition, subtraction, multiplication and division) and logical (AND, OR, XOR and shift) operations. Also, Integer arithmetic unit calculates the target address for unconditional or conditional jump and branch instructions. Integer execution unit executes the system related instruction such as SCALL, SBREAK instructions for supervisor level access of the instruction. An efficient data forwarding scheme is used to forward output of execution units to the input of the execution unit. Detailed micro-architecture for data forwarding scheme is explained in section VI.

Unit for Memory related instructions calculate the target data memory address for load and store operations. RISC-V ISA supports load or store operations on a byte, half-word and word data to and from data memory. Micro-architecture of an out-of-order FP execution unit is explained in section V.

E. Write Back Stage

Write Back (WB) stage commits the instruction from the pipeline and updates the register file with the results from execute units. WB reads the instruction from top of the scheduled instruction FIFO and based on the pipeline information, it reads either from integer or memory or floating point concurrent units.

V. MICRO ARCHITECTURE DESIGN OF OUT-OF ORDER FLOATING POINT UNIT

Fig6 depicts the top level architecture of floating point co-processor. Instruction decoder partially decodes the floating point instructions, after register select operations, the function fields with op-code are passed to the out-of-order co-processor. Floating point data operands to the co-processor are selected from the register select unit. Write back stage accepts the responses from co-processor and updates the destination register. To make the use of out-of-order, functional modules are developed individually, integrated as a single module with specific token for instruction scheduling.

A. FPU request

Input request packet contains operand (192-bit) for floating point operation, op-code and function fields. Fig7 shows the FPU request packet format. Each FPU request is stored in input FIFO of depth 2 which is further read by input decoder. Input operand width is fixed to 192-bit which holds 3 data operands, either a 32-bit (single precision or integer) or 64-bit (double precision) floating point.

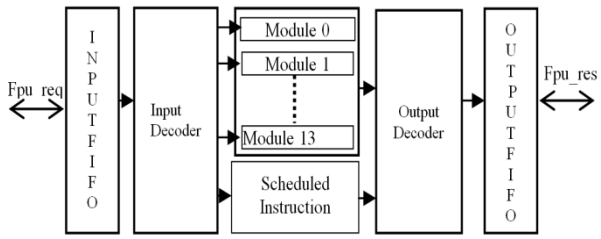

Fig 6: Architecture of Out-of-order FPU Co-processor

0……..…191	0…….6	0…….6	0.....4	0 1 2	0 1 2
Operand(SP/DP)	Op code	F7	F5	F3	RM

Fig 7: FPU request packet

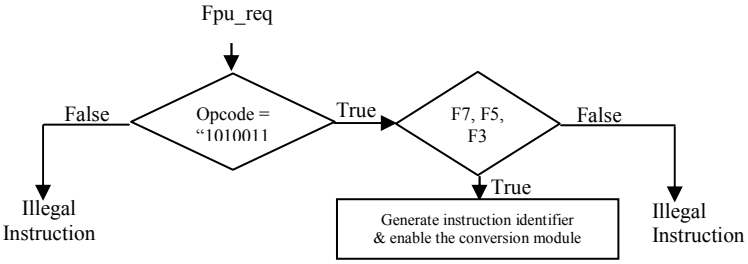

Fig 8: Flow diagram of Input decoder

Input request packet contains operand (192-bit) for floating point operation, op-code and function fields. Fig7 shows the FPU request packet format. Each FPU request is stored in input FIFO of depth 2 which is further read by input decoder. Input operand width is fixed to 192-bit which holds 3 data operands, either a 32-bit (single precision or integer) or 64-bit (double precision) floating point.

B. Input Decoder

Fig8 depicts the flow diagram of input decoder. Input Decoder in floating point pipeline reads the FPU request packet from the Input FIFO, decodes the floating point operation based on the op-code, F7, F5 and F3 fields. After decoding, corresponding individual hardware module is enabled and instruction token for the specific module is stored in the scheduled instruction FIFO. Illegal floating point operations are tagged with illegal token and stored in the scheduled instruction FIFO

C. Scheduled Instruction FIFO

Scheduled instruction FIFO stores the instruction tokens, which specifies present operations in the pipeline. Scheduled instruction FIFO has a depth of 6, which is equal to the maximum number of pipeline stages in the co-processor.

D. Functional Units

Functional units are for performing various floating point operations. Various FPU instructions are decoded to perform out-of-order execution in these functional units. Addition and subtraction operations share the same functional unit pipeline and have a clock latency of 5. Conversion operations have a latency of 5 clock cycles. Multiplication, division and square root operations have a clock cycle latency of 11, 31 and 34 clock cycles respectively. Operation results and exceptions are raised according to the IEEE 754-2008 standard.

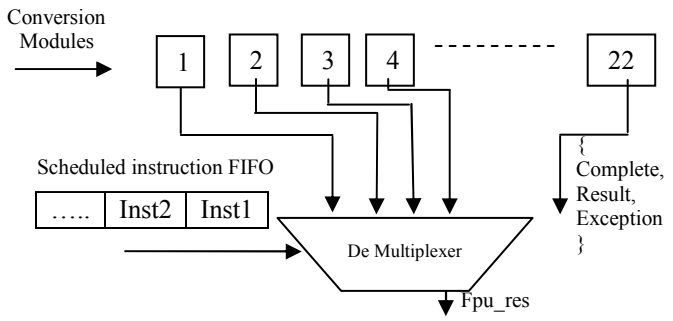

Fig 9: Flow diagram of Output decoder

E. Output Decoder

Fig 9 shows the flow diagram of output decoder. Output decoder commits the oldest instruction from pipeline. To ensure in-order commit, output Decoder in the pipeline reads the scheduled instruction from the top of the scheduled instruction FIFO and polls the completion signal from the matched individual module. Once the floating point operation is completed, the result (32-bit or 64-bit), exceptions (5-bit) from the module is read and stored in output FIFO which has a depth of 2. On completion of instruction execution, first token from the top of the instruction scheduler FIFO is removed.

F. FPU response

Output FPU response packet holds maximum 64-bit result which can be of either 32-bit (single precision floating point or integer) or 64-bit (double precision floating point) result. Exceptions from each individual module are collected and stored in floating point Control and status registers.

VI. Design Issues and Options in RISC-V Pipeline

Proposed core is designed by considering various aspects like hazard, grouping of instructions, decoder logic complexity, register file access, and pipeline operations. RISC-V pipeline is associated with two hazards; data hazards and control hazards.

A. Data hazards

Data hazard (Read-After-Write (RAW)) in RISC-V pipeline occurred due to data dependency between instructions. RISC-V decoder stage has two 32-bit registers (named as scoreboard) keep the usage status of integer and floating point registers by the instructions in the pipeline by means of destination register address. In decode stage, an instruction will get decoded and destination register position will get updated in specified scoreboard. When an instruction completes its operation at the write back stage, corresponding bit position in scoreboard is reset. Performance impact of data hazard (RAW hazard) on pipeline stage is handled by data forwarding scheme. Data forwarding scheme is applicable only in case if both current and previous instructions are of integer type.

Data dependency in pipeline stages can be determined with 2-scoreboard registers and current instructions; pipeline access (integer, memory, FPU), register access, and register update information (instruction information). Output of data hazard detection unit specifies whether, integer execution output have to forward the data to input. Data forwarding unit is a combinational unit. Data hazard in RISC-V pipeline such as

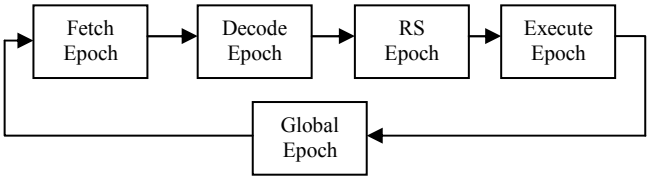

Fig 10: Epoch Register Mechanism

RAW, WAW, WAR are handled in pipeline by either forwarding the data for RAW or by introducing bubbles in the pipeline for WAW hazard. Since, proposed core is in-order commit WAR hazard will not occur in our RISC-V pipeline.

B. Control hazards

Control hazard alias branch hazard causes a greater performance loss in RISC-V pipeline than data hazard. RISC-V ISA has unconditional jump and conditional branch instructions, causes clock cycle penalty during branch miss prediction. In our RISC-V pipeline, prediction is micro-architected by using sophisticated branch predication scheme and miss- prediction is micro-architected using EPOCH register mechanism.

1. Branch Miss Prediction

In RISC-V pipeline the branch target address is calculated in execute stage, meanwhile three successive instructions from branch instructions are entered into the pipeline, which have to be flushed or not based on the outcome of branch instruction. For jump instructions the pipeline needs to be flushed, which leads to a minimum penalty of 3 clock cycles. RISC-V branch instructions are conditional, if the branch condition is satisfied causes a penalty of 3 clock cycles and the pipeline has to be flushed. Epoch register mechanism is used to flush the instructions in processor pipeline. Basically the meaning of epoch is storing the history of something. Proposed processor contains a global epoch register common to all pipelines stages along with local epoch registers for each pipeline stage.

Epoch registers are used with single bit granularity. Fig 10 shows the orientation of Epoch register in each pipeline stage. The mechanism of epoch register concept is as follows: Initially all the epoch registers are initialized with logic zero. Since, the stages in the pipeline are sequential, instructions pass stage by stage. Along with the instructions, the global epoch value is circulated from one stage to the next stage till execute stage. When a branch instruction is encountered in pipeline, the target address of that particular branch is calculated in execute stage. In execute stage; the predicted PC is compared with the target address. If the target address does not match with the predicted PC, the global epoch register value is inverted and the PC is updated with the target address. Therefore, for the first miss prediction, global epoch becomes logic one. In the next clock cycle, the global epoch and local epoch registers of execute stage are compared. If the local epoch register value in execute stage mismatches with the global epoch register, the instruction will be discarded.

2. Branch Prediction

Clock cycle penalty caused by flushing the pipeline is reduced by implementing a branch prediction scheme. Fig 2 shows the integration of Branch predictor to the fetch unit. A sophisticated branch prediction of [6] [7] [8] scheme is adopted in our processor to realize Branch prediction and Branch Target Buffer.

C. Grouping of instruction& Decode Level logic

RISC-V instructions are categorized based on the functionality and op-code (instruction [6:0]). Base instruction set (RV32) is classified into 8 categories which are elaborated in section III. Decoder uses fixed levels of different comparators to identify an instruction. Proposed decoder is organized in such a way that system instructions are having maximum level of logic with 7, 3, and 2-bit comparators

D. Register File Access and pipeline operations

RISC-V ISA supports, both integer and floating point registers. Operand accessed from either integer or floating point register file.It is determined at decoder stage. In effect, the register file access requires additional combinational module (multiplexer) for selecting the operand access.

VII. Verification and Analysis

A. Verification of core

Integer and floating pipelines of RISC-V Instruction Set Processor are developed by using Bluespec System Verilog (BSV), integrated with FPGAs Block RAM which serves as L1 cache. Floating point pipeline is verified at test bench level using pseudo random test vectors. For pseudo- random verification at test bench level, the input test vectors are generated using a high level language ("C") and applied to the Device Under Test (DUT) and also to the Matlab tool. The Matlab tool generated outputs are considered as the golden output and are compared against the outputs from the DUT. Core pipeline is verified with hex file generated using hand coded assembled instructions.

B. Implementation on FPGA

Processor core isprimarilysynthesized on Xilinx Virtex 6 "xc6vlx550t-2ff1759" FPGA. Table II depicts the devices utilization of proposed core on FPGA, open RISC and Rocket core (RISC-V implementation from UCB).

TABLE II: Device Utilization of processor core on FPGA

Parameter	Proposed Core (with FPU)	Open RISC [17] (without FPU)	Rocket Core [18] (with Limited FPU ops)
Slice Registers	18340	2280	12388
Slice LUTs	46530	6744	46256
LUT-FF pair	11759	7063	5425
DSP48E1s	32	4	22

TABLE III: Synthesis results of core – 65nm and 130nm

Parameter/Technology Node		65 nm		130 nm	
		LL @ 500MHz	SP @ 500MHz	LL @ 500MHz	SP@ 500MHz
Logic Cells	Proposed (with FPU)	173 K	171 K	1424 K	1287 K
	Rocket Core [18] (with Limited FPU ops)	121 K	121 K	1328 K	1178 K
Area	Proposed	561 mm^2	557 mm^2	2539 mm^2	2408 mm^2
	Rocket Core[18] (with Limited FPU ops)	451 mm^2	465 mm^2	2118 mm^2	1824 mm^2

C. Implementation on ASIC

Processor has been synthesized using Cadence Encounter RTL compiler using UMC 65nm and 130 nm with Low Leakage (LL), Standard Performance (SP), and High Performance (HP) libraries. Table III depicts the device of proposed core, open RISC and Rocket core.

Our instruction set processor consumes more hardware resource which is attributed to, that it is with independent hardware for FP add/sub, multiply, divide, square-root etc., out-of-order execute, in-order commit, deeply pipelined FP arithmetic units which improves throughput of FPU operations.

VIII. Conclusions

A 5-satge pipelined 32-bit instruction set processor compatible with RISC-V ISA has been micro-architected and analyzed the design issues / options for pipeline stages in terms ofgrouping of instruction, decoder logic complexity, register file access and control flow instructions. Processor includes a sophisticated branch prediction and advanced data hazard detection with data forwarding unit and IEEE 754-2008 compliant out-of-order execute and in-order-commit FPU.Both integer and floating point pipeline are verified at test bench level,core is implemented on Xilinx Virtex 6xc6vlx550t-2ff1759 FPGA and ASIC (UMC 65 and 130 nm library).

IX. Future works

In future, the core has been proposed to be enhanced with more rigorous data forwarding schemes between pipeline stages, concurrent load/store operations with load/store queue, and cache controller. With such features it is planned to test and verify the core with compiler generated code.

Acknowledgements

We express our sincere gratitude to Prof. AmruturBharadwaj, Department of Electrical Communication Engineering, Indian Institute of Science (IISc), Bangalore for helping us in synthesizing the core on various technology nodes.

REFERENCES

[1] A. Waterman, Y. Lee, David A. Patterson, KrsteAsanovi The RISC-V Instruction Set Manual,Vol. I: Base User-Level ISA. Tech. Rep. UCB/EECS-2011-62, EECS Department, UCB, May 2011.

[2] OpenRISC 1000 Arch. Manual, 1st Ed., OpenCores, 2012.

[3] Clerc S. Crolles, Abouzeid, F. Patel, D.A. Daveau, J.-M. Bottoni, C. Ciampolini L, Giner F, "Design and Performance Parameters of an Ultra-Low Voltage, Single Supply 32-bit Processor implemented in 28nm FDSOI Technology" Proc. of IEEE Intl. Symp.on Quality Electronic Design, pp. 366-370, 2015.

[4] Lopez-Parrado, A.Valderrama-Cuervo,"OpenRISC-based System-on-Chip for Digital Signal Processing" Proc. of IEEE symposium on Image, signal processing and Artificial Vision, pp. 1-5, Sept 2014.

[5] AndreyMokhov, Iliasov, Sokolov, Rykunov, Yakovlev, Romanovsky"Synthesis of Processor Instruction Sets from High-Level ISA Specifications", IEEE trans on computers, Vol:63, pp. 1551-1565, June 2014

[6] Liney Group article "The case of open instruction sets, open ISA would enable free competition in processor design". August 2014

[7] Krste Asanović & David Patterson, UC Berkeley, EE Times article " RISC-V Open standard for SoCs, The case of Open ISA" July 2014

[8] Krste Asanović David A. Patterson "Instruction Sets Should Be Free: The Case For RISC-V" 2014

[9] Andreas Olofsson "Analyzing the RISC-V Instruction Set Architecture" August 2014.

[10] User Manual "Epiphany Architecture Reference"

[11] Open source Processor Development "lowRISC" 2015

[12] Shakthi Processor series IIT Madras 2015

[13] T-Y Yeh and Y.N. Patt, "Two-Level Adaptive Branch Prediction", Technical Report CSE-TR-117-91, Computer Science & Engineering Division, Department of EECS, University of Michigan, (Nov. 1991).

[14] Chang, P.-Y., Maris Evers, Y.N. Patt "Improving branch prediction accuracy by reducing pattern history table interference", 1996 International Conference on Parallel architecture and compilation techniques, pp. 48-57, 1996

[15] IEEE standards Board and ANSI. IEEE Standards for Binary Floating-point Arithmetic, 2008, IEEE Std., 754-2008.

[16] J. Hennessy and D. D. Patterson, "Computer architecture, a quantitative approach". Morgan Kaufman, 2003.

[17] Damjan Lampret et al., "OpenRISC 1000 Architecture Manual", Architecture Version 1.0, Document Revision 0, December 5, 2012

[18] RISC-V processor core "Rocket Core" EECS Department, UCB, May 2011.