Computer Organization
Floating Point Coprocessor

Instruction Set Architecture

Guillermo Briceño
Rene Nicklich
Austin Lantz

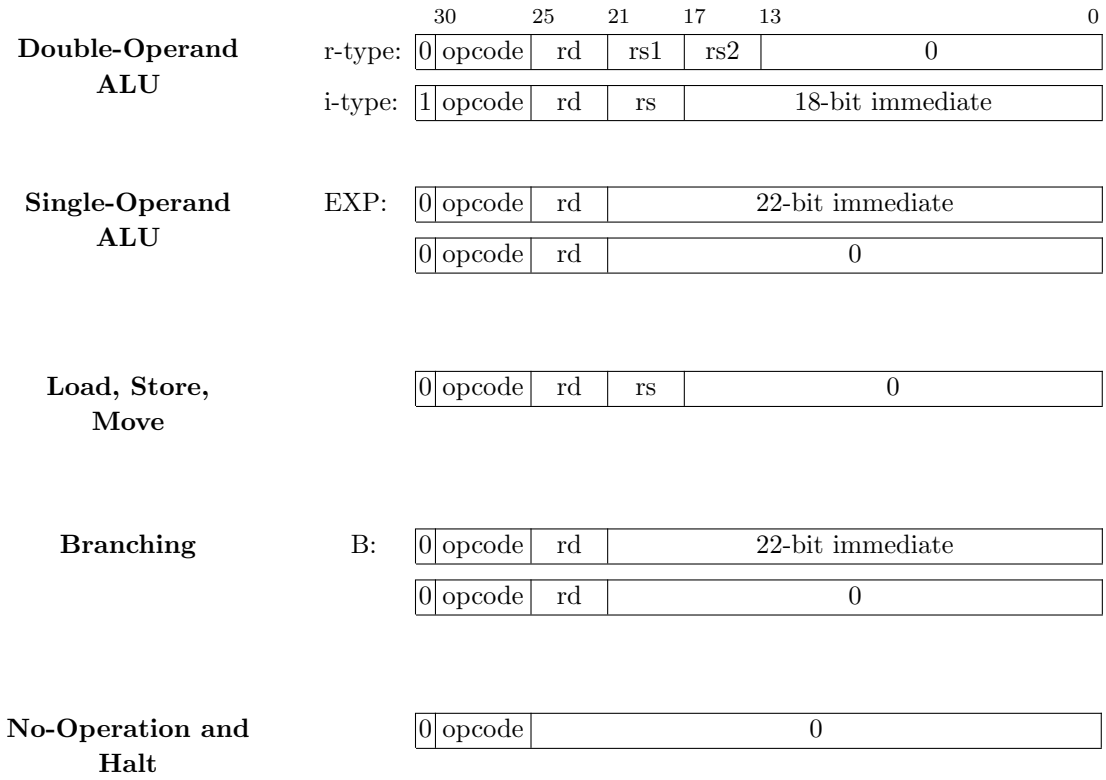
February 28, 2019

1 Introduction

This document describes the instruction set architecture for a 32-bit floating-point coprocessor. Each instruction is described with its assembler syntax in section 2.

1.1 Instruction Encoding

The twenty-three instructions can be grouped by their instruction encoding formats. The keywords `rd`, `rs1`, and `rs2` stand for the 4-bit register addresses from $\texttt{0000}_2$ to $\texttt{1111}_2$, corresponding to a destination register, the first source register, and second source register respectively. In single-source or immediate mode addressing, the keyword `rs` is used instead. Bit 31 is used to specify register or immediate addressing mode for double-operand ALU instructions.



2 Instruction Set Specification

NOP

No operation

Opcode:	`00000`
Syntax:	`NOP`
Purpose:	Perform no operations.

SET

Set register to floating-point value

Opcode:	`00001`
Syntax r-type:	`SET rd, #<32-bit FP value>`
Purpose:	Assign a 32-bit floating point value to rd.
Operation:	rd ← FPvalue

LOAD

Load value from memory

Opcode:	`00010`
Syntax r-type:	`LOAD rd, rs`
Purpose:	Assign rd the value from the memory address in rs.
Operation:	rd ← M[rs]

STORE

Store value to memory

Opcode:	`00011`
Syntax r-type:	`STORE rd, rs`
Purpose:	Assign memory location specified in rd to value in rs.
Operation:	M[rd] ← rs

MOVE

Copy value from a register to another

Opcode:	`00100`
Syntax r-type:	`MOVE rd, rs`
Purpose:	Assign rd the value in rs.
Operation:	rd ← rs

FADD

Add 32-bit floating-point value

Opcode:	`00101`
Syntax r-type:	`FADD rd, rs1, rs2`
Syntax i-type:	`FADD rd, rs1, #<32-bit ?  immediate>`
Purpose:	Performs addition on two 32-bit floating-point values from rs1 and rs2, or an immediate in place of rs2, and stores the result in rd.
Operation:	rd ← rs1 + rs2 or rd ← rs1 + immediate
Condition Codes:	C N Z V x x x x

FSUB

Subtract 32-bit floating-point values

Opcode:	`00110`
Syntax r-type:	`FSUB rd, rs1, rs2`
Syntax i-type:	`FSUB rd, rs1, #<32-bit ?  immediate>`
Purpose:	Performs subtraction on two 32-bit floating-point values from rs1 and rs2, or an immediate in place of rs2, and stores the result in rd.
Operation:	rd ← rs1 - rs2 or rd ← rs1 - immediate
Condition Codes:	C N Z V x x x x

FNEG

Negate a 32-bit floating-point value

Opcode:	`00111`
Syntax:	`FNEG rd, rs`
Purpose:	Performs negation on a 32-bit floating-point value from rs and stores the result in rd.
Operation:	rd ← -rs
Condition Codes:	C N Z V - x x -

FMUL

Multiply two 32-bit floating-point values

Opcode:	`01000`
Syntax r-type:	`FMUL rd, rs1, rs2`
Syntax i-type:	`FMUL rd, rs1, #<32-bit ? immediate>`
Purpose:	Performs multiplication on two 32-bit floating-point values from rs1 and rs2, or an immediate in place of rs2, and stores the result in rd.
Operation:	rd $\leftarrow$ rs1 $*$ rs2 or rd $\leftarrow$ rs1 $*$ immediate
Condition Codes:	C N Z V x x x x

FDIV

Divide two 32-bit floating-point values

Opcode:	`01001`
Syntax r-type:	`FDIV rd, rs1, rs2`
Syntax i-type:	`FDIV rd, rs1, #<32-bit ? immediate>`
Purpose:	Performs division on two 32-bit floating-point values from rs1 and rs2, or an immediate in place of rs2, and stores the result in rd.
Operation:	rd $\leftarrow$ rs1 $\div$ rs2 or rd $\leftarrow$ rs1 $\div$ immediate
Condition Codes:	C N Z V x x x x

FLOOR

Compute the floor function

Opcode:	`01010`
Syntax r-type:	`FLOOR rd, rs`
Purpose:	Rounds the value in rs to the nearest lowest integer and stores the result in rd.
Operation:	rd $\leftarrow \lfloor$rs$\rfloor$
Condition Codes:	C N Z V - x x -

CEIL

Compute the ceiling function

Opcode:	`01011`
Syntax:	`CEIL rd, rs`
Purpose:	Rounds the value in rs to the nearest highest integer and stores the result in rd.
Operation:	rd $\leftarrow \lceil$rs$\rceil$
Condition Codes:	C N Z V - x x -

ROUND

Round a value

Opcode:	`01100`
Syntax:	`ROUND rd, rs`
Purpose:	Rounds the value in rs and stores the result in rd.
Operation:	rd $\leftarrow$ round(rs)
Condition Codes:	C N Z V - x x x

FABS

Compute the absolute value

Opcode:	`01101`
Syntax:	`FABS rd, rs`
Purpose:	Find the absolute value in rs and stores the result in rd.
Operation:	rd $\leftarrow$ |rs|
Condition Codes:	C N Z V - - x -

MIN

Find the smallest value

Opcode:	`01110`
Syntax:	`MIN rd, rs1, rs2`
Purpose:	Finds the smallest value between rs1 and rs2 and stores the result in rd.
Operation:	rd $\leftarrow$ min(rs1, rs2)
Condition Codes:	C N Z V - x - -

MAX

Find the largest

Opcode:	`01111`
Syntax:	`MAX rd, rs1, rs2`
Purpose:	Finds the largest value between rs1 and rs2 and store the result in rd.
Operation:	rd ← max(rs1, rs2)
Condition Codes:	C N Z V - x - -

POW

Compute the power

Opcode:	`10000`
Syntax:	`POW rd, rs, #<integer-value>`
Purpose:	Finds rs1 raised to an integer value and stores the result in rd.
Operation:	rd ← $rs^{\text{integer-value}}$
Condition Codes:	C N Z V x x x x

EXP

Compute the exponent

Opcode:	`10001`
Syntax:	`EXP rd, rs`
Purpose:	Finds e raised to the value in rs and stores the result in rd.
Operation:	rd ← e^{rs}
Condition Codes:	C N Z V x x x x

SQRT

Compute the square root of a value

Opcode:	`10010`
Syntax:	`SQRT rd, rs`
Purpose:	Finds the square root of the value in rs and stores the result in rd.
Operation:	rd $\leftarrow \sqrt{\text{rs}}$
Condition Codes:	C N Z V - x - -

B

Branch unconditionally

Opcode:	`10011`
Syntax:	`B rd`
Purpose:	Set the program counter to the value in memory addressed by rd.
Operation:	PC $\leftarrow$ M[rd]

BZ

Branch if zero

Opcode:	`10100`
Syntax:	`BZ rd, <LABEL>`
Purpose:	Branch to the label specified in the assembly program if rd is equal to zero.
Operation:	if (rd == 0): PC $\leftarrow$ LABEL

BN

Branch if negative

Opcode:	`10101`
Syntax:	`BN rd, <LABEL>`
Purpose:	Branch to the label specified in the assembly program if rd is less than zero.
Operation:	if (rd $<$ 0): PC $\leftarrow$ LABEL

HALT

Stop program

Opcode:	`10110`
Syntax:	`HALT`
Purpose:	Stop the program.