Overview

CS01 is a thirty-two-bit processor modelled after the RISCV ISA (RV32I). Only a subset of the RISCV instruction set is implemented. The processor features 32, 32-bit integer registers and 32, 32-bit floating-point registers. The processor has the most prominent features from real processors with some of the more complex details left out.

Programming Model

	Registers			
	31 0		31 0	
	x0 / zero		f0 / zero	
	x1 / ra		f1	
	x2 / fp		…	
	x3-x13 / s1-s11		…	
	x14 / sp		…	
	x15 / tp		…	
	x16-x17 / v0-v1		…	
	x18-x25 / a0-a7		…	
	x26-x30 / t0 – t4		f30	
	x31 / gp		f31	
	pc			

x? refers to an integer register. f? refers to a floating-point register

Registers x0 and f0 are always zero. f0 is positive zero.

x14 / sp is the stack pointer.

pc is the program counter.

Nomenclature

The size of values are referred to as the following:

Size	Alternate	
1	byte	byte
2	wyde	half-word
4	tetra	word
5	penta	
8	octa	double-word
10	deci	
16	hexi	

Instruction Set Formats

Bits	31 25	24 20	19 15	14 12	11 7	6 0	
LUI[2]	$imm_{31..12}$				Rd	55	
AUIPC	$imm_{31..12}$				Rd	23	
JAL	20 $imm_{10..1}$ 11		$imm_{19..12}$		Rd	111	
CALL[1]	20 $imm_{10..1}$ 11		$imm_{19..12}$		1	111	
JALR	$imm_{11..0}$		Rs1	0	Rd	103	
RET[1]	0		1	0	0	103	
BEQ	$imm_{12,10..5}$	Rs2	Rs1	0	$imm_{4..1,11}$	99	
BNE	$imm_{12,10..5}$	Rs2	Rs1	1	$imm_{4..1,11}$	99	
BLT	$imm_{12,10..5}$	Rs2	Rs1	4	$imm_{4..1,11}$	99	
BGE	$imm_{12,10..5}$	Rs2	Rs1	5	$imm_{4..1,11}$	99	
BLTU	$imm_{12,10..5}$	Rs2	Rs1	6	$imm_{4..1,11}$	99	
BGEU	$imm_{12,10..5}$	Rs2	Rs1	7	$imm_{4..1,11}$	99	
BRA[1]	$imm_{12,10..5}$	0	0	0	$imm_{4..1,11}$	99	
LB	$imm_{11..0}$		Rs1	0	Rd	3	LDB
LH	$imm_{11..0}$		Rs1	1	Rd	3	LDW
LW	$imm_{11..0}$		Rs1	2	Rd	3	LDT
LBU	$imm_{11..0}$		Rs1	4	Rd	3	LDBU
LHU	$imm_{11..0}$		Rs1	5	Rd	3	LDWU
FLW	$imm_{11..0}$		Rs1	2	FRd	7	LDFT
SB	$imm_{11..5}$	Rs2	Rs1	0	$Imm_{4..0}$	35	STB
SH	$imm_{11..5}$	Rs2	Rs1	1	$Imm_{4..0}$	35	STW
SW	$imm_{11..5}$	Rs2	Rs1	2	$Imm_{4..0}$	35	STT
FSW	$imm_{11..5}$	FRs2	Rs1	2	$Imm_{4..0}$	39	STFT
ADDI	$imm_{11..0}$		Rs1	0	Rd	19	
NOP[1]	0		0	0	0	19	
SLTI	$imm_{11..0}$		Rs1	2	Rd	19	
SLTUI	$imm_{11..0}$		Rs1	3	Rd	19	
XORI	$imm_{11..0}$		Rs1	4	Rd	19	EORI
ORI	$imm_{11..0}$		Rs1	6	Rd	19	
LDI[1]	$imm_{11..0}$		0	6	Rd	19	
ANDI	$imm_{11..0}$		Rs1	7	Rd	19	
SLLI	0	shamt	Rs1	1	Rd	19	SHLI
SRLI	0	shamt	Rs1	5	Rd	19	SHRI
SRAI	16	shamt	Rs1	5	Rd	19	ASRI
ADD	0	Rs2	Rs1	0	Rd	51	
SUB	32	Rs2	Rs1	0	Rd	51	
MUL	1	Rs2	Rs1	0	Rd	51	
SLL	0	Rs2	Rs1	1	Rd	51	SHL
SLT	0	Rs2	Rs1	2	Rd	51	
SLTU	0	Rs2	Rs1	3	Rd	51	
XOR	0	Rs2	Rs1	4	Rd	51	EOR
SRL	0	Rs2	Rs1	5	Rd	51	SHR
SRA	32	Rs2	Rs1	5	Rd	51	ASR
OR	0	Rs2	Rs1	6	Rd	51	
MOV[1]	0	0	Rs1	6	Rd	51	
AND	0	Rs2	Rs1	7	Rd	51	
FADD	0 00	FRs2	FRs1	rm	FRd	83	
FSUB	1 00	FRs2	FRs1	rm	FRd	83	
FMUL	2 00	FRs2	FRs1	rm	FRd	83	
FDIV	3 00	FRs2	FRs1	rm	FRd	83	
FMIN	5 00	FRs2	FRs1	0	FRd	83	

FMAX	5	00	FRs2	FRs1	1	FRd	83	
FSQRT	11	00	0	FRs1	rm	FRd	83	
FSGNJ	16	00	FRs2	FRs1	0	FRd	83	
FMOV[1]	16	00	FRs1	FRs1	0	FRd	83	
FSGNJN	16	00	FRs2	FRs1	1	FRd	83	
FNEG[1]	16	00	FRs1	FRs1	1	FRd	83	
FSGNJX	16	00	FRs2	FRs1	2	FRd	83	
FABS[1]	16	00	FRs1	FRs1	2	FRd	83	
FEQ	20	00	FRs2	FRs1	2	Rd	83	
FLT	20	00	FRs2	FRs1	1	Rd	83	
FLE	20	00	FRs2	FRs1	0	Rd	83	
FCVT.W.S	24	00	0	FRs1	rm	Rd	83	
FCVT.WU.S	24	00	1	FRs1	rm	Rd	83	
FCVT.S.W	25	00	0	Rs1	rm	FRd	83	
FCVT.S.WU	25	00	1	Rs1	rm	FRd	83	
FMV.X.S	28	00	0	FRs1	0	Rd	83	
FCLASS	28	00	0	FRs1	1	Rd	83	
FMV.S.X	30	00	0	Rs1	0	FRd	83	
FENCE	0	pred	succ	0	0	0	15	
FENCE.I	0	0	0	0	1	0	15	
ECALL	000h			0	0	0	115	
EBREAK	001h			0	0	0	115	
ERET	100h			0	0	0	115	
RDCYCLE	C00h			0	1	Rd	115	
RDCYCLEH	C80h			0	1	Rd	115	
RDTIME	C01h			0	1	Rd	115	
RDTIMEH	C81h			0	1	Rd	115	
RDINSTRET	C02h			0	1	Rd	115	
RDINSTRETH	C82h			0	1	Rd	115	
WFI	101h			0	0	0	115	
PFI[2]	103h			0	0	0	115	

1. an extended mnemonic for another instruction
2. instruction is a green-field extension to the RISCV instruction set

Integer Instructions

ADD – Addition

Description:

Add two values using two's complement addition, which are in Rs1 and Rs2 or an immediate value and place the sum in the destination register Rd.

Instruction Format: R2, RI

Exceptions: none

AND – Bitwise And

Description:

Bitwise 'and' two values which are in Rs1 and Rs2 or an immediate value and place the result in the destination register Rd. A bitwise operation operates on each bit of the register individually. By carefully managing values the bitwise and may also be used as a logical and.

Instruction Format: R2, RI

Exceptions: none

AUIPC – Add Upper Immediate to PC

Description:

This instruction adds the upper 20 bits of the program counter to an immediate supplied by the instruction and stores the result in the destination register Rd. This instruction may be used to generate addresses relative to the program counter.

EOR – Bitwise Exclusive Or

Description:

This is an alternate mnemonic supported by the assembler for the XOR instruction. Bitwise ‘exclusive or’ two values which are in Rs1 and Rs2 or an immediate value and place the result in the destination register Rd. A bitwise operation operates on each bit of the register individually. By carefully managing values the bitwise eor may also be used as a logical eor.

Instruction Format: R2, RI

Exceptions: none

LDI – Load Immediate

Description:

This is an alternate mnemonic for the ‘or’ instruction where Rs1 is assumed to be x0. This has the effect of simply loading the constant into integer register Rd.

Instruction Format: RI

Exceptions: none

LUI – Load Upper Immediate

Description:

The LUI instruction sets the upper 20 bits of the destination register Rd to the constant supplied in the instruction and zeros out the lower 12 bits of the destination register.

MOV – Move Register

Description:

This is an alternate mnemonic for the ‘or’ instruction where Rs2 is assumed to be x0. The value in Rs1 is then simply copied to destination register Rd.

Instruction Format: R2

Exceptions: none

OR – Bitwise Inclusive Or

Description:

Bitwise ‘inclusive or’ two values which are in Rs1 and Rs2 or an immediate value and place the result in the destination register Rd. A bitwise operation operates on each bit of the register individually. By carefully managing values the bitwise or may also be used as a logical or.

Instruction Format: R2, RI

Exceptions: none

SLL – Shift Left Logical

Description:

Shift left the value in Rs1 by the value in Rs2 or an immediate value and place the result in the destination register Rd. Low order bits are filled with zeros.

Instruction Format: R2, RI

Exceptions: none

SLT – Set if Less Than

Description:

Compare two two's complement signed values which are in Rs1 and Rs2 or an immediate value. If Rs1 is less than the second operand then store a one in register Rd, otherwise store a zero in register Rd.

Instruction Format: R2, RI

Exceptions: none

SRA – Shift Right Arithmetic

Description:

Shift right the value in Rs1 by the value in Rs2 or an immediate value and place the result in the destination register Rd. High order bits are filled with the original sign bit, preserving the sign of the number.

Instruction Format: R2, RI

Exceptions: none

SRL – Shift Right Logical

Description:

Shift right the value in Rs1 by the value in Rs2 or an immediate value and place the result in the destination register Rd. High order bits are filled with zeros.

Instruction Format: R2, RI

Exceptions: none

XOR – Bitwise Exclusive Or

Description:

Bitwise ‘exclusive or’ two values which are in Rs1 and Rs2 or an immediate value and place the result in the destination register Rd. A bitwise operation operates on each bit of the register individually. By carefully managing values the bitwise xor may also be used as a logical xor.

Instruction Format: R2, RI

Exceptions: none

Control Flow Instructions

BEQ – Branch if Equal

Description:

This instruction tests if two registers are equal and branches if they are; otherwise program execution continues with the next instruction. The branch target is calculated as the sum of the program counter and a sign extended displacement value found in the instruction.

Instruction Format: BCC

Exceptions: none

BGE – Branch if Greater Than or Equal

Description:

This instruction tests if two registers and branches if Rs1 is greater than or equal to Rs2; otherwise program execution continues with the next instruction. The values in registers Rs1 and Rs2 are treated as two's complement signed numbers. The branch target is calculated as the sum of the program counter and a sign extended displacement value found in the instruction.

Instruction Format: BCC

Exceptions: none

BGEU – Branch if Greater Than or Equal Unsigned

Description:

This instruction tests if two registers and branches if Rs1 is greater than or equal to Rs2; otherwise program execution continues with the next instruction. The values in registers Rs1 and Rs2 are treated as unsigned numbers. The branch target is calculated as the sum of the program counter and a sign extended displacement value found in the instruction.

Instruction Format: BCC

Exceptions: none

BLT – Branch if Less Than

Description:

This instruction tests if two registers and branches if Rs1 is less than Rs2; otherwise program execution continues with the next instruction. The values in registers Rs1 and Rs2 are treated as two's complement signed numbers. The branch target is calculated as the sum of the program counter and a sign extended displacement value found in the instruction.

Instruction Format: BCC

Exceptions: none

BLTU – Branch if Less Than Unsigned

Description:

This instruction tests if two registers and branches if Rs1 is less than Rs2; otherwise program execution continues with the next instruction. The values in registers Rs1 and Rs2 are treated as unsigned numbers. The branch target is calculated as the sum of the program counter and a sign extended displacement value found in the instruction.

Instruction Format: BCC

Exceptions: none

BNE – Branch if Not Equal

Description:

This instruction tests if two registers are unequal and branches if they are; otherwise program execution continues with the next instruction. The branch target is calculated as the sum of the program counter and a sign extended displacement value found in the instruction.

Instruction Format: BCC

Exceptions: none

BRA – Branch Always

Description:

This instruction is an alternate mnemonic for the BEQ instruction where both registers are assumed to be x0. Hence the branch is always taken. The branch target is calculated as the sum of the program counter and a sign extended displacement value found in the instruction.

Instruction Format: BRA

Exceptions: none

CALL – Call Subroutine

Description:

This is an alternate mnemonic for the JAL instruction where the destination register is assumed to be the $ra register.

Instruction Format: JAL, JALR

Exceptions: none

FENCE[.I]

Description:

With this core the fence instruction is a nop operation. Memory instructions are not buffered and always execute in order. Fencing is used to control order on machines where the order of memory operation may not be in program order.

JAL – Jump and Link

Description:

The JAL instruction jumps to the target address determined by adding a signed extended immediate constant in the instruction to the program counter. The constant is shifted left once before the addition. The two LSB's of the target address are set to zero. The address of the next instruction after the JAL is stored in the destination register Rd. The address range of the JAL instruction is approximately +/- 1 MB.

Instruction Format: JAL

Exceptions: none

JALR – Jump and Link Register

Description:

The JALR instruction jumps to the target address determined by adding a signed extended immediate constant in the instruction to integer register Rs1. The two LSB's of the target address are set to zero. The address of the next instruction after the JALR is stored in the destination register Rd. The address range is all of memory.

Instruction Format: JALR

Exceptions: none

RET – Return from Subroutine

Description:

RET is an alternate mnemonic for the JALR instruction where the constant is assumed to be zero and the source register is the return address register x1. The RET instruction is common to many instruction sets. Another mnemonic for this instruction is RTS.

Instruction Format: JALR

Exceptions: none

Memory Instructions

Address Modes

The processor supports only a single address mode – register indirect with displacement. Any other desired addressing of data must be built up out of instructions using this address mode.

Unaligned Accesses

If there is an unaligned access for data larger than a byte, the processor will automatically run two bus cycles to load or store the data. The processor doesn't care what address is used for the data; however, using aligned accesses results in faster program execution as only single bus cycles are required.

FLW – Float Load Word (32 bits)

Description:

FLW loads 32-bit data from memory and loads it into the floating-point destination register Rd. The memory address to load from is calculated as the sum of integer register Rs1 and an immediate constant in the instruction.

Instruction Format: ML

Exceptions: none

FSW – Float Store Word (32 bits)

Description:

FSW stores 32-bit data to memory from the floating-point destination source register Rs2. The memory address to store to is calculated as the sum of integer register Rs1 and an immediate constant in the instruction.

Instruction Format: MS

Exceptions: none

LB – Load Byte (8 bits)

Description:

LB loads a byte of data from memory, sign extends it to the width of the machine, and loads it into the integer destination register Rd. The memory address to load from is calculated as the sum of Rs1 and an immediate constant in the instruction.

Instruction Format: ML

Exceptions: none

LBU – Load Byte Unsigned (8 bits)

Description:

LBU loads a byte of data from memory, zero extends it to the width of the machine, and loads it into the integer destination register Rd. The memory address to load from is calculated as the sum of Rs1 and an immediate constant in the instruction.

Instruction Format: ML

Exceptions: none

LDB – Load Byte (8 bits)

Description:

LDB is an alternate mnemonic for the LB instruction. It loads a byte of data from memory, sign extends it to the width of the machine, and loads it into the integer destination register Rd. The memory address to load from is calculated as the sum of Rs1 and an immediate constant in the instruction.

Instruction Format: ML

Exceptions: none

LDBU – Load Byte Unsigned (8 bits)

Description:

LDBU is an alternate mnemonic for LBU. LDBU loads a byte of data from memory, zero extends it to the width of the machine, and loads it into the integer destination register Rd. The memory address to load from is calculated as the sum of Rs1 and an immediate constant in the instruction.

Instruction Format: ML

Exceptions: none

LDT – Load Tetra (32 bits)

Description:

LDT is an alternate mnemonic for the LW instruction which loads 32-bit data from memory and loads it into the integer destination register Rd. The memory address to load from is calculated as the sum of Rs1 and an immediate constant in the instruction.

Instruction Format: ML

Exceptions: none

LDW – Load Wyde (16 bits)

Description:

LDW is an alternate mnemonic for the LH instruction. LDW loads 16-bit data from memory, sign extends it to the width of the machine, and loads it into the integer destination register Rd. The memory address to load from is calculated as the sum of Rs1 and an immediate constant in the instruction.

Instruction Format: ML

Exceptions: none

LDWU – Load Wyde Unsigned (16 bits)

Description:

LDWU is an alternate mnemonic for LHU. LHU loads 16-bit data from memory, zero extends it to the width of the machine, and loads it into the integer destination register Rd. The memory address to load from is calculated as the sum of Rs1 and an immediate constant in the instruction.

Instruction Format: ML

Exceptions: none

LH – Load Half (16 bits)

Description:

LH loads 16-bit data from memory, sign extends it to the width of the machine, and loads it into the integer destination register Rd. The memory address to load from is calculated as the sum of Rs1 and an immediate constant in the instruction.

Instruction Format: ML

Exceptions: none

LHU – Load Half Unsigned (16 bits)

Description:

LHU loads 16-bit data from memory, zero extends it to the width of the machine, and loads it into the integer destination register Rd. The memory address to load from is calculated as the sum of Rs1 and an immediate constant in the instruction.

Instruction Format: ML

Exceptions: none

LW – Load Word (32 bits)

Description:

LW loads 32-bit data from memory and loads it into the integer destination register Rd. The memory address to load from is calculated as the sum of Rs1 and an immediate constant in the instruction.

Instruction Format: ML

Exceptions: none

SB – Store Byte (8 bits)

Description:

SB stores a byte of data to memory from the low order eight bits of source register Rs2. The memory address to load from is calculated as the sum of Rs1 and an immediate constant in the instruction.

Instruction Format: MS

Exceptions: none

SH – Store Half (16 bits)

Description:

SH stores 16-bits of data to memory from the low order sixteen bits of source register Rs2. The memory address to load from is calculated as the sum of Rs1 and an immediate constant in the instruction.

Instruction Format: MS

Exceptions: none

STB – Store Byte (8 bits)

Description:

STB is an alternate mnemonic of the SB instruction. SB stores a byte of data to memory from the low order eight bits of source register Rs2. The memory address to load from is calculated as the sum of Rs1 and an immediate constant in the instruction.

Instruction Format: MS

Exceptions: none

STT – Store Tetra (32 bits)

Description:

STT is an alternate mnemonic for the SW instruction. SW stores 32-bits of data to memory from source register Rs2. The memory address to load from is calculated as the sum of Rs1 and an immediate constant in the instruction.

Instruction Format: MS

Exceptions: none

STW – Store Wyde (16 bits)

Description:

STW is an alternate mnemonic of the SH instruction. SH stores 16-bits of data to memory from the low order sixteen bits of source register Rs2. The memory address to load from is calculated as the sum of Rs1 and an immediate constant in the instruction.

Instruction Format: MS

Exceptions: none

SW – Store Word (32 bits)

Description:

SW stores 32-bits of data to memory from source register Rs2. The memory address to load from is calculated as the sum of Rs1 and an immediate constant in the instruction.

Instruction Format: MS

Exceptions: none

Floating-Point

Rounding mode

The rounding mode to use for floating point instructions may be one specified in the instruction or a dynamic rounding mode specified in the rounding mode register. If the rounding mode specified in the instruction is ‘111’ then the dynamic rounding mode register will be used to determine the rounding mode.

Floating-Point Exceptions

Underflow occurs when the result is a de-normal number having an exponent of zero. Underflow sets the uf bit in the floating-point status register.

Overflow occurs when the result becomes infinite (positive or negative); the exponents is all ones and the mantissa is zero. Overflow sets the of bit in the floating-point status register.

Inexact occurs during normalization if there were bits in the intermediate result that were non-zero to the right of the LSB of the result. Inexact sets the nx bit in the floating-point status register.

Divide by zero occurs if an attempt is made to divide a number by zero or an attempt is made to take the square root of zero. Divide by zero sets the dz bit in the floating-point status register.

Invalid operation occurs if there is an attempt to take the square root of a negative number. An invalid operation sets the nv bit in the floating-point status register.

FABS – Absolute Value

Description:

FABS is an alternate mnemonic for FSGNJX which copies the value in Rs1 into the destination register Rd then sets the sign of Rd equal to the xor of the sign of Rs1 and Rs2. Rs1 and Rs2 are encoded as the same register by the assembler.

Instruction Format: FSGNJ

Exceptions: none

FADD – Addition

Description:

FADD adds two floating-point values in floating-point registers Rs1 and Rs2 and store the result in floating-point register Rd. If either operand is a Nan then the result is a Nan.

Instruction Format: FLT

Exceptions: uf, of, nx

FDIV – Division

Description:

FDIV divides two floating-point values in floating-point registers Rs1 and Rs2 and stores the result in floating-point register Rd. If either operand is a Nan then the result is a Nan.

Instruction Format: FLT

Exceptions: uf, of, nx, dz

FEQ – Float Test for Equality

Description:

This instruction tests two floating-point values in registers Rs1 and Rs2 for equality. If the condition is true a one is returned in integer register Rd. Rs1 and Rs2 are floating-point registers. Positive zero and negative zero are assumed to be equal. If either operand is a Nan, then this test will return false.

Instruction Format: FLT

Exceptions: none

FLE – Float Test for Less Than or Equal

Description:

This instruction tests two floating-point values in registers Rs1 and Rs2 for Rs1 less than or equal to Rs2. If Rs1 is less than or equal to Rs2 then Rd is set to one. Otherwise Rd is set to zero. Rd is an integer register, Rs1 and Rs2 are floating-point registers. Positive zero and negative zero are assumed to be equal. If either operand is a Nan, then this test will return false. This instruction may also be used to test for greater than or equal by swapping the operands.

Instruction Format: FLT

Exceptions: none

FLT – Float Test for Less Than

Description:

This instruction tests two floating-point values in registers Rs1 and Rs2 for Rs1 less than Rs2. If Rs1 is less than Rs2 then Rd is set to one. Otherwise Rd is set to zero. Rd is an integer register, Rs1 and Rs2 are floating-point registers. Positive zero and negative zero are assumed to be equal. If either operand is a Nan, then this test will return false. This instruction may also be used to test for greater than by swapping the operands.

Instruction Format: FLT

Exceptions: none

FMOV – Move Register

Description:

FMOV is an alternate mnemonic for FSGNJ which copies the value in Rs1 into the destination register Rd then sets the sign of Rd equal to the sign of Rs2. Rs1 and Rs2 are encoded as the same register by the assembler. Both the source and destination registers are part of the floating-point register file. To move directly between the integer and floating-point register files see the FMV instruction.

Instruction Format: FSGNJ

Exceptions: none

FMUL – Multiplication

Description:

FMUL multiplies two floating-point values in floating-point registers Rs1 and Rs2 and stores the result in floating-point register Rd. If either operand is a Nan then the result is a Nan.

Instruction Format: FLT

Exceptions: uf, of, nx

FMV – Move Register

Description:

The FMV instruction moves a value directly between integer and floating-point registers without performing any conversions. FMV.X.S moves from a floating-point register Rs1 to an integer register Rd. FMV.S.X moves an integer register Rs1 to a floating-point register Rd.

Instruction Format: FMV

Exceptions: none

FNEG – Negate

Description:

FNEG is an alternate mnemonic for FSGNJN which copies the value in Rs1 into the destination register Rd then sets the sign of Rd equal to the complement of the sign of Rs2. Rs1 and Rs2 are encoded as the same register by the assembler.

Instruction Format: FSGNJ

Exceptions: none

FSGNJ – Sign Injection

Description:

FSGNJ copies the value in Rs1 into the destination register Rd then sets the sign of Rd equal to the sign of Rs2.

Instruction Format: FSGNJ

Exceptions: none

FSGNJN – Sign Injection Invert

Description:

FSGNJ copies the value in Rs1 into the destination register Rd then sets the sign of Rd equal to the complement of the sign of Rs2.

Instruction Format: FSGNJ

Exceptions: none

FSGNJX – Sign Injection Xor

Description:

FSGNJX copies the value in Rs1 into the destination register Rd then sets the sign of Rd equal to the xor of the sign of Rs1 and Rs2.

Instruction Format: FSGNJ

Exceptions: none

FSUB – Subtraction

Description:

FSUB subtracts two floating-point values in floating-point registers Rs1 and Rs2 and stores the result in floating-point register Rd. If either operand is a Nan then the result is a Nan.

Instruction Format: FLT

Exceptions: uf, of, nx

Machine Mode Instructions

EBREAK – Debug Environment Call

Description:

This instruction transfers control back to the debug environment. The processor is switched to machine mode with interrupts disabled. An ERET instruction should be used to return from an environment call.

ECALL – Environment Call

Description:

This instruction invokes environment (operating system) processing. The processor is switched to machine mode with interrupts disabled. An ERET instruction should be used to return from an environment call.

ERET – Return from Exception

Description:

This instruction returns to user mode from an exception handler. The previous interrupt mask setting is restored.

PFI – Poll for Interrupt

Description:

This instruction causes the processor to check for the presence of an interrupt then perform interrupt processing if an interrupt is present. Otherwise program execution continues with the next instruction. Interrupts do no have to be enabled for the PFI instruction to perform interrupt processing. Effectively PFI temporarily enables interrupts for the duration of the instruction.

Green-Field Extension

This instruction is a green-field extension to the base RISCV instruction set and not likely to be present in other implementations. An equivalent action may be performed using a minimum sequence of two CSR instructions to enable then disable interrupts.

Instruction Format: PFI

Exceptions: none

WFI – Wait for Interrupt

Description:

This instruction causes the processor to pause and wait for an interrupt signal before continuing. While waiting for an interrupt the processor clock is stopped to reduce power consumption. Only the wall-clock time is updated. If an interrupt occurs and interrupts are enabled, then the interrupt service routine will begin. Otherwise if an interrupt occurs and interrupts are not enabled, then program execution will continue with the next instruction.

Instruction Format: WFI

Exceptions: none

Write a program to load two numbers into x1 and x2 and store the result in x3.

Write a program to compute the rom checksum. The rom checksum is the sum of all the bytes in the rom.

Compute the clocks per instruction using the tick count and instructions retired CSR registers. Compute the result using floating point instructions.

Where does the processor go when system mode is entered?

> The processor continues on from where it was last before entering user mode. Since user mode is entered with an eret instruction, the address is the next address after the eret. However, at reset the processor begins running in system mode at the reset address of $FFFC0000.

Reset

> The RISCV spec pretty much leaves it up to the implementor to set the reset address. There are generally two used areas for the reset address, a high address or a low address. Ram memory often begins at a low address, so the author chose a high address for the reset address. A small rom is placed at $FFFC0000 in the upper range of addresses. Often the rom contains just enough code to load an OS into memory from a I/O device such as disk, or memory card.

Machine Mode Programming Model

Machine mode has its own integer register file.

	Registers	
	31 0	
	x0 / zero	
	x1 / ra	
	x2 / fp	
	x3-x13 / s1-s11	
	x14 / sp	
	x15 / tp	
	x16-x17 / v0-v1	
	x18-x25 / a0-a7	
	x26-x30 / t0 – t4	
	x31 / gp	
	pc	

x? refers to an integer register. f? refers to a floating-point register

Registers x0 and f0 are always zero. f0 is positive zero.

x14 / sp is the stack pointer.

pc is the program counter.

System Argument Registers

System argument registers are a set of non-standard CSR registers used to communicate between user and machine mode. They are CSR's 0x800 to 0x807.

Reset Operation

On reset the processor begins executing instructions at $FFFC0100 in machine mode. Interrupts are disabled. All other state is undefined.