Assembly Language Lecture Series: RV32D instructions: Double precision floating point

Roger Luis Uy

College of Computer Studies

De La Salle University

Manila, Philippines

RV32D instructions

RV32D Load and store instructions	
FLD	FSD

RV32D Computational instructions				
FADD	FSUB	FMUL	FDIV	FSQRT
FMIN	FMAX	FSQRT	FMADD	FMSUB

RV32D move instructions				
FSGNJ	FSGNJN	FSGNJX	FMV.X.D	FMV.D.X

RV32D Conversion instructions				
FCVT.W.D	FCVT.WU.D	FCVT.D.W	FCVT.D.WU	
FCVT.S.D	FCVT.D.S			

RV32D comparison instructions				
FEQ	FLT	FLE		

RV32D classify instruction	
FCLASS	

RV32D Instructions

Double-Precision Registers

RISC-V floating point registers

- Floating point register for double-precision is 64-bit
- Double-precision registers can hold either 32-bit or 64-bit floating-point
- When multiple floating-point precisions are supported, then valid values of narrower 32-bit types are represented in the lower 32 bits of a 64-bit NaN value, in a process termed NaN-boxing.
- The upper bits of a valid NaN-boxed value must be all 1s.

RISC-V floating-point registers

Register name	Symbolic name	Description	Owner
f0-f7	ft0-ft7	Floating-point temporaries	Caller
f8-f9	fs0-fs1	Floating-point saved registers	Callee
f10-f11	fa0-fa1	Floating-point arguments/return values	Caller
f12-f17	fa2-fa7	Floating-point arguments	Caller
f18-f27	s2-fs11	Floating-point saved registers	Callee
f28-f31	ft8-ft11	Floating-point temporaries	caller

Register name	Symbolic name	Description	Owner
Fcsr		Floating-point control and status register	

Floating-point control status register

31-8	7-5	4-0				
Reserved	Rounding mode (**frm**)	Accrued exceptions (**fflags**)				
24	3	NV	DZ	OF	UF	NX

Rounding mode	Mnemonic	Meaning
000	RNE	Round to nearest, ties to even
001	RTZ	Round towards zero (truncate)
010	RDN	Round down
011	RUP	Round up
100	RMM	Round to nearest, ties to max magnitude
101		Invalid. Reserve for future use
110		Invalid. Reserve for future use
111	DYN	In instruction's rm field, selects dynamic rounding mode; In Rounding Mode register, Invalid.

Flag Mnemonic	Flag meaning
NV	Invalid operation
DZ	Divide by zero
OF	Overflow
UF	Underflow
NX	Inexact

Floating-point control status register (CSR)

- The *fcsr* register can be read and written with the FRCSR and FSCSR instructions
- FRCSR reads *fcsr* by copying it into integer register *rd*.
- FSCSR swaps the value in *fcsr* by copying the original value into integer register *rd*, and then writing a new value obtained from integer register *rs*1 into *fcsr*.
- The FRRM instruction reads the Rounding Mode field *frm* and copies it into the least-significant three bits of integer register *rd*, with zero in all other bits.
- FSRM swaps the value in *frm* by copying the original value into integer register *rd*, and then writing a new value obtained from the three least-significant bits of integer register *rs* into *frm*.
- FRFLAGS and FSFLAGS are defined for the Accrued Exception Flags field fflags.

Floating-point CSR pseudo-instruction

Pseudo-instruction	Base instruction	Description
frcsr *rd*	csrrs *rd*, *fcsr*, x0	Read fp control/status register
fscsr *rd*, *rs*	csrrw *rd*, *fcsr*, *rs*	Swap fp control/status register
fcsr *rs*	csrrw x0, *fcsr*, *rs*	Write fp control/status register
frrm *rd*	csrrs *rd*, *rm*, x0	Read fp rounding mode
fsrm *rd*, *rs*	csrrw *rd*, *rm*, *rs*	Swap fp rounding mode
fsrm *rs*	csrrw x0, *rm*, *rs*	Write fp rounding mode
frflags *rd*	csrrs *rd*, *fflags*, x0	Read fp exception flags
fsflags *rd*, *rs*	csrrw *rd*, *fflags*, rs	Swap fp exception flags
fsflags *rs*	csrrw x0, *fflags*, rs	Write fp exception flags

RV32D Instructions

Double-Precision Load/Store Instructions

FLD instruction

FLD *rd*, *offset*(*rs*)

- loads a double-precision value from memory into floating-point register *rd*.
- Effective address is obtained by adding register *rs* to the sign-extended 12-bit offset.
- pseudo-instruction *la*(load address) is used to initialize a register to point to a memory

Example:

```
.data
var1: .double 4.0
.text
la t0, var1
fld f0, 0(t0)
```

After execution:

F0 = 4010000000000000

FSD instruction

FSD *rs2*, *offset*(*rs1*)

- stores a double-precision value from floating-point register *rs2* to memory.
- Effective address is obtained by adding register *rs1* to the sign-extended 12-bit offset.
- pseudo-instruction *la*(load address) is used to initialize a register to point to a memory

Example:

```
.data
var1: .double 4.0
var2: .double 0.0
.text
la t0, var1
la t1, var2
fld f0, 0(t0)
fsd f0, 0(t1)
```

After execution:

F0 = 4010000000000000

var2 = 4010000000000000

RV32D Instructions

Double-Precision Computational Instructions

FADD/FSUB instruction

FADD.D *rd*, *rs1, rs2*

FSUB.D *rd*, *rs1, rs2*

- FADD.D performs double-precision floating point addition between *rs*1 and *rs*2, result is written in *rd*.
- FSUB.D performs the double-precision floating point subtraction of *rs*2 from *rs*1, result is written in *rd*.

Example:

```
.data
var1: .double 4.0
var2: .double 5.0
.text
la t0, var1
la t1, var2
fld f0, 0(t0)
fld f1, 0(t0)
fadd.d f2, f0, f1
fsub.d f3, f0, f1
```

After execution:

F0 = 4010000000000000 (4.0)

F1 = 4014000000000000 (5.0)

F2 = 4022000000000000 (9.0)

F3 = BFF0000000000000 (-1.0)

FMUL/FDIV instruction

FMUL.D *rd*, *rs1, rs2*

FDIV.D *rd*, *rs1, rs2*

- FMUL.D performs double-precision floating point multiplication between *rs*1 and *rs*2, result is written in *rd*.
- FDIV.D performs double-precision floating-point division of *rs*1 by *rs*2, result is written in *rd*.

Example:

```
.data
var1: .double 4.0
var2: .double 5.0
.text
la t0, var1
la t1, var2
fld f0, 0(t0)
fld f1, 0(t0)
fmul.d f2, f0, f1
fdiv.d f3, f0, f1
```

After execution:

F0 = 4010000000000000 (4.0)

F1 = 4014000000000000 (5.0)

F2 = 4032000000000000 (20.0)

F3 = 3FE999999999999A (0.8)

FSQRT instruction

FSQRT.D *rd*, *rs1*

- FSQRT.D computes the square root of *rs*1, result is written in *rd*.

Example:

```
.data
var1: .double 4.0
.text
la t0, var1
fld f0, 0(t0)
fsqrt.d f2, f0
```

After execution:

F0 = 4010000000000000 (4.0)

F2 = 4000000000000000 (2.0)

FMADD instruction

FMADD.D *rd*, *rs1, rs2, rs3*

- FMADD.D multiplies the values in *rs*1 and *rs*2, adds the value in *rs*3, and writes the final result to *rd*.
- Formula: (*rs*1×*rs*2)+*rs*3.

Example:

```
.data
var1: .double 2.0
var2: .double 3.0
var3: .double 4.0
.text
la t0, var1
la t1, var2
la t2, var3
fld f0, 0(t0)
fld f1, 0(t1)
fld f2, 0(t2)
fmadd.d f3, f0,f1,f2
```

After execution:

F0 = 4010000000000000 (2.0)

F1 = 4008000000000000 (3.0)

F2 = 4010000000000000 (4.0)

F3 = 4024000000000000 (10.0)

FMSUB instruction

FMSUB.D *rd*, *rs1, rs2, rs3*

- FMSUB.D multiplies the values in *rs*1 and *rs*2, subtracts the value in *rs*3, and writes the final result to *rd*.
- Formula: (*rs*1×*rs*2)-*rs*3.

Example:

```
.data
var1: .double 2.0
var2: .double 3.0
var3: .double 4.0
.text
la t0, var1
la t1, var2
la t2, var3
fld f0, 0(t0)
fld f1, 0(t1)
fld f2, 0(t2)
fmsub.d f3, f0,f1,f2
```

After execution:

F0 = 4010000000000000 (2.0)

F1 = 4008000000000000 (3.0)

F2 = 4010000000000000 (4.0)

F3 = 4010000000000000 (2.0)

FMIN/FMAX instruction

FMIN.D *rd*, *rs1, rs2*

FMAX.D *rd*, *rs1, rs2*

- FMIN.D writes the smaller of *rs*1 or *rs*2 to *rd*.
- FMAX.D writes the larger of *rs*1 or *rs*2 to *rd*.

Example:

```
.data
var1: .double 2.0
var2: .double 3.0
.text
la t0, var1
la t1, var2
fld f0, 0(t0)
fld f1, 0(t0)
fmin.d f2, f0, f1
fmax.d f3, f0, f1
```

After execution:

F0 = 4010000000000000 (2.0)

F1 = 4008000000000000 (3.0)

F2 = 4010000000000000 (2.0)

F3 = 4008000000000000 (3.0)

FPMIN/FMAX instructions

- For the purposes of these instructions only, the value –0.0 is considered to be less than the value +0.0
- If both inputs are NaNs, the result is the canonical NaN.
- If only one operand is a NaN, the result is the non-NaN operand.

RV32D Instructions

Double-Precision Move Instructions

FMSGNJ/N/X instruction

FSGNJ.D *rd*, *rs1, rs2*

FSGNJN.D *rd*, *rs1, rs2*

FSGNJX.D *rd*, *rs1*, *rs2*

- FSGNJ.D (Sign inject), FSGNJN.D, and FSGNJX.D, produce a result that takes all bits except the sign bit from *rs1*.
  - For FSGNJ, the result's sign bit is *rs2*'s sign bit
  - for FSGNJN, the result's sign bit is the opposite of *rs2*'s sign bit
  - FSGNJX, the sign bit is the XOR of the sign bits of *rs1* and *rs2*.

Example:

```
.data
var1: .double -4.0
var2: .double 5.0
.text
la t0, var1
la t1, var2
fld f0, 0(t0)
fld f0, 0(t1)
fsgnj.d f2, f0, f1
fsgnjn.d f3, f0, f1
fsgnjx.d f4, f0, f1
```

After execution:

F0 = C010000000000000 (-4.0)

F1 = 4014000000000000 (5.0)

F2 = 4010000000000000 (+4.0)

F3 = C010000000000000 (-4.0)

F4 = C010000000000000 (-4.0)

Some useful FSGNJ pseudo-instructions

Pseudo-instruction	Base instruction	Description
fmv.d *rd*, *rs*	Fsgn.d *rd*, *rs*, *rs*	Double precision register to register transfer
fabs.*d rd, rs*	Fsgnjx.d *rd*, *rs*, *rs*	Double precision absolute value
fneg.d *rd, rs*	Fsgnjn.d *rd*, *rs*, *rs*	Double precision negate

FMV.D.X/FMV.X.D instruction

FMV.D.X *fd*, *rs1*

FMV.X.D *rd*, *fs*

- FMV.D.X moves the double-precision value encoded in IEEE 754-2008 standard encoding from the integer register *rs1* to the floating-point register *fd*. The bits are not modified in the transfer.
- FMV.X.D moves the double-precision value from the floating-point register *fs* to the integer register *rd*. The bits are not modified in the transfer.

Example:

```
.data
var1: .double 4.0
.text
la t0, var1
fld f0, 0(t0)
fmv.x.d x10, f0
```

After execution:

f0 = 4010000000000000

x10 = 4010000000000000

Example:

```
.data
var1: .dword 0x4010000000000000
.text
la t0, var1
ld x10, 0(t0)
fmv.d.x f0, x10
```

After execution:

f0 = 4010000000000000

x10 = 4010000000000000

RV32D Instructions

Double-Precision Convert Instructions

FCVT.W.D/FCVT.L.D instruction

FCVT.W.D *rd*, *fs*

FCVT.L.D *rd*, *fs*

- FCVT.W.D converts a double precision in *fs* to a 32-bit signed integer in *rd*.
- FCVT.L.D converts a double precision in *fs* to a 64-bit signed integer in *rd*.

Example:

```
.data
var1: .double -6.0
.text
la t0, var1
fld f0, (t0)
fcvt.w.d x10, f0
fcvt.l.d x12, f0
```

After execution:

f0 = C018000000000000 (-6.0)

x10 = FFFFFFFA (-6)

x12 = FFFFFFFFFFFFFFFA (-6)

FCVT.WU.D/FCVT.LU.D instruction

FCVT.WU.D *rd*, *fs*

FCVT.LU.D *rd*, *fs*

- FCVT.WU.D converts a double precision in *fs* to a 32-bit unsigned integer in *rd*.
- FCVT.LU.D converts a double precision in *fs* to a 64-bit unsigned integer in *rd*.

Example:

```
.data
var1: .double 4.0
.text
la t0, var1
fld f0, (t0)
fcvt.wu.d x2, f0
fcvt.lu.d x4, f0
```

After execution:

f0 = 4010000000000000 (4.0)

x2 = 00000004

x4 = 0000000000000004

FCVT.D.W/FCVT.D.L instruction

FCVT.D.W *fd*, *rs*

FCVT.D.L *fd*, *rs*

- FCVT.D.W converts a 32-bit signed integer in *rs* to double precision floating point in *fd*.
- FCVT.D.L converts a 64-bit signed integer in *rs* to double precision floating point in *fd*.

Example:

```
.data
var1: .word -4
var2: .dword -4
.text
la t0, var1
lw x10, (t0)
fcvt.d.w f0, x10
la t1, var2
ld x12, (t1)
fcvt.d.l f2, x12
```

After execution:

f0 = C010000000000000 (-4.0)

f2 = C010000000000000 (-4.0)

x10 = FFFFFFFC

x12 = FFFFFFFFFFFFFFFC

FCVT.D.WU/FCVT.D.LU instruction

FCVT.D.WU *fd*, *rs*

FCVT.D.LU *fd*, *rs*

- FCVT.D.WU converts a 32-bit unsigned integer in *rs* to double precision floating point in *fd*.
- FCVT.D.LU converts a 64-bit unsigned integer in *rs* to double precision floating point in *fd*.

Example:

```
.data
var1: .word -4
var2: .dword -4 (0xFFFFFFFFFFFFFFFC)
.text
la t0, var1
lw x10, (t0)
fcvt.d.wu f0, x10
la t1, var2
ld x12, (t1)
fcvt.d.lu f2, x12
```

After execution:

x10 = FFFFFFFC

f0 = 41efffffff800000 (4294967292)

x12 = FFFFFFFFFFFFFFFC

f2 = 43f0000000000000 (18446744073709551614)

FCVT.S.D instruction

FCVT.S.D *fd*, *fs*

- FCVT.S.D converts a double precision in *fs* to single precision in *fd*.
- FCVT.S.D rounds according to the *rm* field

Example:

```
.data
var1: .double 5.5
.text
la t0, var1
fld f0, (t0)
fcvt.s.d f1, f0
```

After execution:

f0 = 4016000000000000 (5.5)

f2 = 40B00000 (5.5)

FCVT.D.S instruction

FCVT.D.S *fd*, *fs*

- FCVT.D.S converts a single precision in *fs* to double precision in *fd*.
- FCVT.D.S will never round

Example:

```
.data
var1: .float 5.5
.text
la t0, var1
flw f0, (t0)
fcvt.d.s f1, f0
```

After execution:

f0 = 40B00000 (5.5)

f1 = 4016000000000000 (5.5)

RV32D Instructions

Double-Precision Comparison Instructions

FEQ/FLT/FLE instruction

FEQ.D *rd*, *fs1, fs2*

FLT.D *rd*, *fs1, fs2*

FLE.D *rd*, *fs1, fs2*

- Floating-point compare instructions (FEQ.D, FLT.D, FLE.D) perform the specified comparison between floating-point registers (*fs1* == *fs2*, *fs1* < *fs2*, *fs1* ≤ *fs2*) writing 1 to the integer register *rd* if the condition holds, and 0 otherwise.

Example:

```
.data
var1: .double 5.0
var2: .double 6.0
.text
la t0, var1
la t1, var2
fld f0, (t0)
fld f1, (t1)
flt.d x10, f0, f1
```

After execution:

x10 = 00000001

f0 = 4014000000000000

f1 = 4018000000000000

x10 = 0000000000000001

RV32D Instructions

Double-Precision Classification Instruction

FCLASS instruction

FCLASS.D *rd*, *fs*

- FCLASS.D instruction examines the value in floating-point register *fs* and writes to integer register *rd* a 10-bit mask that indicates the class of the floating-point number.
- The corresponding bit in *rd* will be set if the property is true and clear otherwise.
- All other bits in *rd* are cleared.
- Note that exactly one bit in *rd* will be set.

Example:

```
.data
var1: .double -5.0
.text
la t0, var1
fld f0, (t0)
fclass.d x10, f0
```

After execution:

x10 = 00000002 (00 0000 0010)

f0 = C014000000000000 (-5.0)

FP Classify instruction

Rd bit	Meaning
0	*fs* is –infinity
1	*fs* is negative normal number
2	*fs* is negative subnormal number
3	*fs* is -0
4	*fs* is +0
5	*fs* is a positive subnormal number
6	*fs* is a positive normal number
7	*fs* is +infinity
8	*fs* is a signaling NaN
9	*fs* is a quiet NaN