Assembly Language Lecture Series: RV32F instructions: Single Precision floating point

Roger Luis Uy

College of Computer Studies

De La Salle University

Manila, Philippines

RV32F instructions

RV32F Load and store instructions	
FLW	FSW

RV32F Computational instructions				
FADD	FSUB	FMUL	FDIV	FSQRT
FMIN	FMAX	FSQRT	FMADD	FMSUB

RV32F move instructions				
FSGNJ	FSGNJN	FSGNJX	FMV.X.W	FMV.W.X

RV32F Conversion instructions				
FCVT.W.S	FCVT.WU.S	FCVT.S.W	FCVT.S.WU	

RV32F comparison instructions				
FEQ	FLT	FLE		

RV32F classify instruction	
FCLASS	

RV32F Instructions

Single-Precision Registers

RISC-V floating point registers

- Floating point register (32-bit for single-precision)
- Floating-point control and status register *fcsr*, which contains the operating mode and exception status of the floating-point unit.

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

Canonical NaN

- If the result of a floating-point operation is NaN, it is called the canonical NaN.
- The canonical NaN has a positive sign and all significand bits clear except the MSB, (a.k.a. the quiet bit).
- For single-precision floating-point, this corresponds to the pattern 0x7FC00000.

Sign bit	Exponent representation	fraction
0	1111 1111	100 0000 0000 0000 0000 0000

RV32F Instructions

Single-Precision Load/Store Instructions

FLW instruction

FLW *rd*, *offset*(*rs*)

- loads a single-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: .float 4.0
.text
la t0, var1
flw f0, 0(t0)
```

After execution:

F0 = 40800000

FSW instruction

FSW *rs2*, *offset*(*rs1*)

- stores a single-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: .float 4.0
var2: .float 0.0
.text
la t0, var1
la t1, var2
flw f0, 0(t0)
fsw f0, 0(t1)
```

After execution:

F0 = 40800000

var2 = 40800000

RV32F Instructions

Single-Precision Computational Instructions

FADD/FSUB instruction

FADD.S *rd*, *rs1, rs2*

FSUB.S *rd*, *rs1, rs2*

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

Example:

```
.data
var1: .float 4.0
var2: .float 5.0
.text
la t0, var1
la t1, var2
flw f0, 0(t0)
flw f1, 0(t1)
fadd.s f2, f0, f1
fsub.s f3, f0, f1
```

After execution:

F0 = 40800000

F1 = 40A00000

F2 = 41100000 (9.0)

F3 = BF800000 (-1.0)

FMUL/FDIV instruction

FMUL.S *rd*, *rs1, rs2*

FDIV.S *rd*, *rs1, rs2*

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

Example:

```
.data
var1: .float 4.0
var2: .float 5.0
.text
la t0, var1
la t1, var2
flw f0, 0(t0)
flw f1, 0(t1)
fmul.s f2, f0, f1
fdiv.s f3, f0, f1
```

After execution:

F0 = 40800000

F1 = 40A00000

F2 = 41A00000 (20.0)

F3 = 3F4CCCCD (0.8)

FSQRT instruction

FSQRT.S *rd*, *rs1*

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

Example:

```
.data
var1: .float 4.0
.text
la t0, var1
flw f0, 0(t0)
fsqrt.s f2, f0
```

After execution:

F0 = 40800000

F2 = 40000000 (2.0)

FMADD.S instruction

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

- FMADD.S 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: .float 2.0
var2: .float 3.0
var3: .float 4.0
.text
la t0, var1
la t1, var2
la t2, var3
flw f0, 0(t0)
flw f1, 0(t1)
flw f2, 0(t2)
fmadd.s f3, f0,f1,f2
```

After execution:

F0 = 40000000 (2.0)

F1 = 40400000 (3.0)

F2 = 40800000 (4.0)

F3 = 41200000 (10.0)

FMSUB.S instruction

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

- FMSUB.S 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: .float 2.0
var2: .float 3.0
var3: .float 4.0
.text
la t0, var1
la t1, var2
la t2, var3
flw f0, 0(t0)
flw f1, 0(t1)
flw f2, 0(t2)
fmsub.s f3, f0,f1,f2
```

After execution:

F0 = 40000000 (2.0)

F1 = 40400000 (3.0)

F2 = 40800000 (4.0)

F3 = 40000000 (2.0)

FMIN/FMAX instruction

FMIN.S *rd*, *rs1, rs2*

FMAX.S *rd*, *rs1, rs2*

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

Example:

```
.data
var1: .float 2.0
var2: .float 3.0
.text
la t0, var1
la t1, var2
flw f0, 0(t0)
flw f1, 0(t1)
fmin.s f2, f0, f1
fmax.s f3, f0, f1
```

After execution:

F0 = 40000000 (2.0)

F1 = 40400000 (3.0)

F2 = 40000000 (2.0)

F3 = 40400000 (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.

RV32F Instructions

Single-Precision Move Instructions

FMSGNJ/N/X instruction

FSGNJ.S *rd*, *rs1, rs2*

FSGNJN.S *rd*, *rs1, rs2*

FSGNJX.S *rd*, *rs1*, *rs2*

- FSGNJ.S (Sign inject), FSGNJN.S, and FSGNJX.S, produce a result that takes all bits except the sign bit from *rs1*.
  - For FSGNJ, replace the sign bit of *rs*1 with sign of *rs*2 and assign it to *rd*.
  - for FSGNJN, replace the sign bit of *rs*1 with the opposite sign of *rs*2 and assign it to *rd*.
  - FSGNJX, XOR the sign bits of *rs1* and *rs2 and assign it to rd.*

Example:

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

After execution:

F0 = C0800000 (-4.0)

F1 = 40A00000 (5.0)

F2 = 40800000 (+4.0)

F3 = C0800000 (-4.0)

F4 = C0800000 (-4.0)

Some useful FSGNJ pseudo-instructions

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

FMV.S.X/FMV.X.S instruction

FMV.S.X *fd*, *rs1*

FMV.X.S *rd*, *fs*

- FMV.S.X moves the single-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.S moves the single-precision value from the floating-point register *fs* to the integer register *rd*. The bits are not modified in the transfer.

Example:

```
.data
var1: .float 4.0
.text
la t0, var1
flw f0, 0(t0)
fmv.x.s x10, f0
```

After execution:

f0 = 40800000 (4.0)

x10 = 40800000

Example:

```
.data
var1: .word 0x40800000
.text
la t0, var1
lw x10, 0(t0)
fmv.s.x f0, x10
```

After execution:

x10 = 40800000

f0 = 40800000 (4.0)

RV32F Instructions

Single-Precision Convert Instructions

FCVT.W.S/FCVT.WU.S instruction

FCVT.W.S *rd*, *fs*

FCVT.WU.S *rd*, *fs*

- FCVT.W.S converts a floating-point number in floating point register *fs* to a signed 32-bit integer and store it in integer register *rd*.
- FCVT.WU.S converts a floating-point number in floating point register *fs* to an unsigned 32-bit integer and store it in integer register *rd*.

Example:

```
.data
var1: .float 4.0
.text
la t0, var1
flw f0, 0(t0)
fcvt.w.s x10, f0
```

After execution:

f0 = 40800000 (4.0)

x10 = 00000004

FCVT.S.W/FCVT.S.WU instruction

FCVT.S.W *fd*, *rs*

FCVT.S.WU *Fd*, *rs*

- FCVT.S.W converts a 32-bit signed integer in integer register *rs* into a floating-point number in floating-point register *fd*.
- FCVT.S.WU converts a 32-bit unsigned integer in integer register *rs* into a floating-point number in floating-point register *fd*.
- A floating-point register can be initialized to floating-point positive zero using FCVT.S.W *fd*, *x0*, which will never set any exception flags.

Example:

```
.data
var1: .word -5
.text
la t0, var1
lw x10, 0(t0)
fcvt.s.w f1, x10
fcvt.s.wu f2, x10
```

After execution:

x10 = FFFFFFFB

f1 = C0A00000 (-5.0)

f2 = 4F800000

(4294967291)

RV32F Instructions

Single-Precision Comparison Instructions

FEQ/FLT/FLE instruction

FEQ.S *rd*, *fs1, fs2*

FLT.S *rd*, *fs1, fs2*

FLE.S *rd*, *fs1, fs2*

- Floating-point compare instructions (FEQ.S, FLT.S, FLE.S) 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: .float 5.0
var2: .float 6.0
.text
la t0, var1
la t1, var2
flw f0, (t0)
flw f1, (t1)
flt.s x10, f0, f1
```

After execution:

x10 = 00000001

f0 = 40A00000 (5.0)

f1 = 40C00000 (6.0)

RV32F Instructions

Single-Precision Classification Instruction

FCLASS instruction

FCLASS.S *rd*, *fs*

- FCLASS.S 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: .float -5.0
.text
la t0, var1
flw f0, (t0)
fclass.s x10, f0
```

After execution:

x10 = 00000002 (00 0000 0010)

f0 = C0A00000 (-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