Information Technology Institute

Ministry of Communications and Information Technology

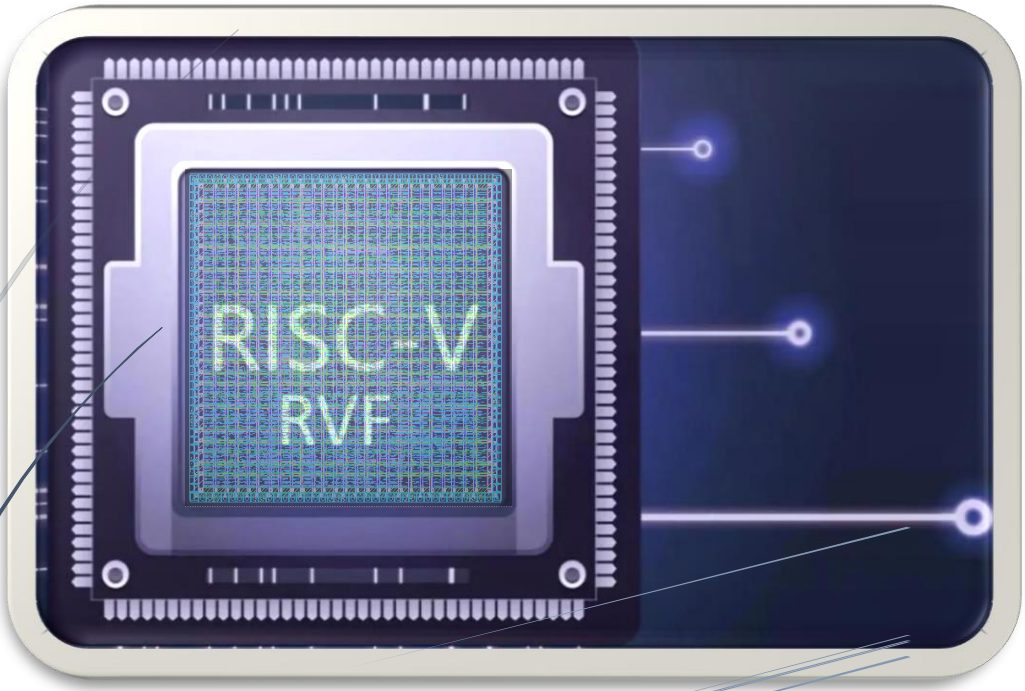
egypt makes electronics

RISC-V Project (Phase 3)

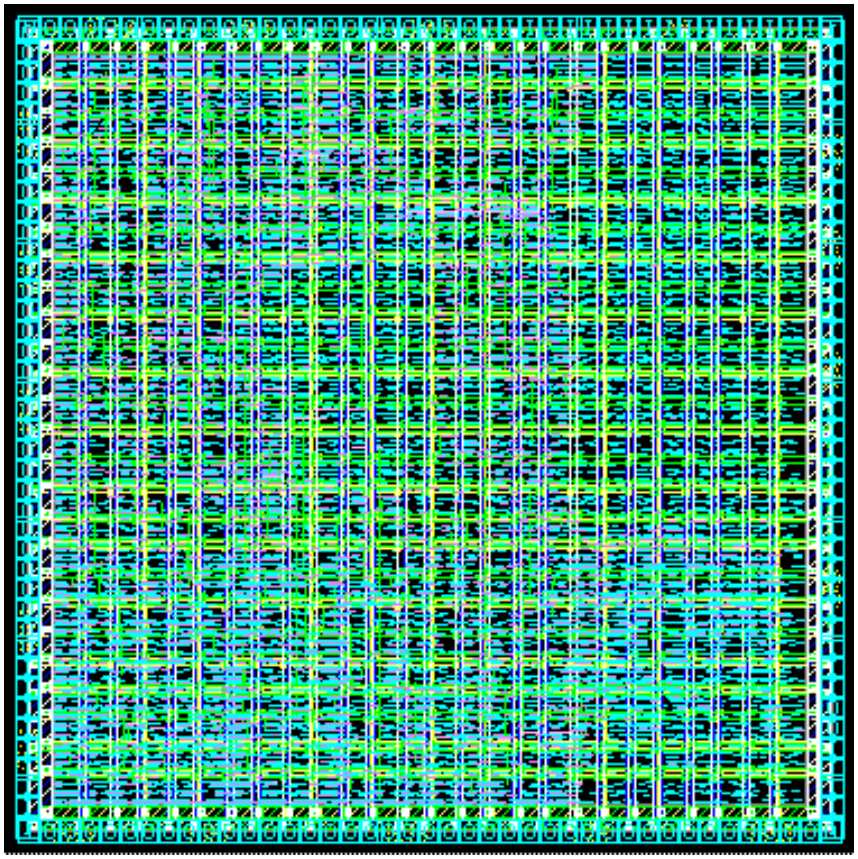
RISC-V (RVF) Floating-Point Extension

Mohamed Elsayed Ali Ebrahim Saad

Digital IC Design (New Capital)



Contents



1. Overview:

❖ RISC-V offers three floating point extensions:

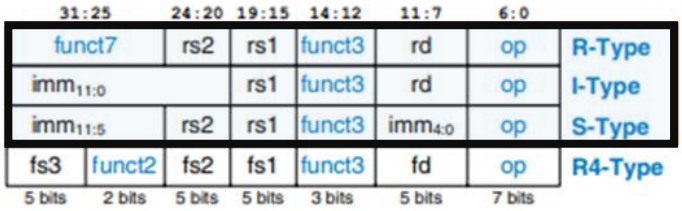
- RVF: single-precision (32-bit): 8 exponent bits, 23 fraction bits.
- RVD: double-precision (64-bit): 11 exponent bits, 52 fraction bits.
- RVQ: quad-precision (128-bit): 15 exponent bits, 112 fraction bits.

- This project describes the standard instruction-set extension for a single-precision floating-point, and single-precision floating-point computational instructions compliant with the IEEE 754-2008 arithmetic standard.

- Floating Point the IEEE 754 standard defines a binary representation for floating point values using three fields.
  - The sign: determines the sign of the number (0 for positive, 1 for negative).
  - The exponent: is in biased notation. For instance, the bias is 127 for single-precision floating point numbers.
  - The significand or mantissa: used to store a fraction instead of an integer.

1	8	23
Sign	Exponent	Mantissa/Significand/Fraction

❖ RVF: RISC-V single-precision floating-point instructions:



op	funct3	funct7	rs2	Type	Instruction	Description	Operation
1000011 (67)	rm	fs3, fmt	–	R4	fmadd fd,fs1,fs2,fs3	multiply-add	fd = fs1 * fs2 + fs3
1000111 (71)	rm	fs3, fmt	–	R4	fmsub fd,fs1,fs2,fs3	multiply-subtract	fd = fs1 * fs2 − fs3
1001011 (75)	rm	fs3, fmt	–	R4	fnmsub fd,fs1,fs2,fs3	negate multiply-add	fd = −(fs1 * fs2 + fs3)
1001111 (79)	rm	fs3, fmt	–	R4	fnmadd fd,fs1,fs2,fs3	negate multiply-subtract	fd = −(fs1 * fs2 - fs3)
1010011 (83)	rm	00000, fmt	–	R	fadd fd,fs1,fs2	add	fd = fs1 + fs2
1010011 (83)	rm	00001, fmt	–	R	fsub fd,fs1,fs2	subtract	fd = fs1 − fs2
1010011 (83)	rm	00010, fmt	–	R	fmul fd,fs1,fs2	multiply	fd = fs1 * fs2
1010011 (83)	rm	00011, fmt	–	R	fdiv fd,fs1,fs2	divide	fd = fs1 / fs2
1010011 (83)	rm	01011, fmt	00000	R	fsqrt fd,fs1	square root	fd = sqrt(fs1)
1010011 (83)	000	00100, fmt	–	R	fsgnj fd,fs1,fs2	sign injection	fd = fs1, sign = sign(fs2)
1010011 (83)	001	00100, fmt	–	R	fsgnjn fd,fs1,fs2	negate sign injection	fd = fs1, sign = −sign(fs2)
1010011 (83)	010	00100, fmt	–	R	fsgnjx fd,fs1,fs2	xor sign injection	fd = fs1, sign = sign(fs2) ^ sign(fs1)
1010011 (83)	000	00101, fmt	–	R	fmin fd,fs1,fs2	min	fd = min(fs1, fs2)
1010011 (83)	001	00101, fmt	–	R	fmax fd,fs1,fs2	max	fd = max(fs1, fs2)
1010011 (83)	010	10100, fmt	–	R	feq rd,fs1,fs2	compare =	rd = (fs1 == fs2)
1010011 (83)	001	10100, fmt	–	R	flt rd,fs1,fs2	compare <	rd = (fs1 < fs2)
1010011 (83)	000	10100, fmt	–	R	fle rd,fs1,fs2	compare ≤	rd = (fs1 ≤ fs2)
1010011 (83)	001	11100, fmt	00000	R	fclass rd,fs1	classify	rd = classification of fs1
RVF only							
0000111 (7)	010	–	–	I	flw fd, imm(rs1)	load float	fd = [Address]31:0
0100111 (39)	010	–	–	S	fsw fs2,imm(rs1)	store float	[Address]31:0 = fd
1010011 (83)	rm	1100000	00000	R	fcvt.w.s rd, fs1	convert to integer	rd = integer(fs1)
1010011 (83)	rm	1100000	00001	R	fcvt.wu.s rd, fs1	convert to unsigned integer	rd = unsigned(fs1)
1010011 (83)	rm	1101000	00000	R	fcvt.s.w fd, rs1	convert int to float	fd = float(rs1)
1010011 (83)	rm	1101000	00001	R	fcvt.s.wu fd, rs1	convert unsigned to float	fd = float(rs1)
1010011 (83)	000	1110000	00000	R	fmv.x.w rd, fs1	move to integer register	rd = fs1
1010011 (83)	000	1111000	00000	R	fmv.w.x fd, rs1	move to f.p. register	fd = rs1

❖ Description for each implemented instruction:

➢ Single-Precision Load and Store Instructions:

Floating-point loads and stores use the same base+offset addressing mode as the integer base.

flw: Load a single-precision floating-point value from memory into floating-point register rd.

31 — 20	19 — 15	14 — 12	11 — 7	6 — 0
imm[11:0]	rs1	width	rd	opcode
12	5	3	5	7
offset[11:0]	base	W	dest	LOAD-FP

fsw: Store a single-precision value from floating-point register rs2 to memory.

31 — 25	24 — 20	19 — 15	14 — 12	11 — 7	6 — 0
imm[11:5]	rs2	rs1	width	imm[4:0]	opcode
7	5	5	3	5	7
offset[11:5]	src	base	W	offset[4:0]	STORE-FP

➢ Single-Precision Floating-Point Conversion and Move Instructions:

➢ All floating-point to integer and integer to floating-point conversion instructions.

31 — 27	26 — 25	24 — 20	19 — 15	14 — 12	11 — 7	6 — 0
funct5	fmt	rs2	rs1	rm	rd	opcode
5	2	5	5	3	5	7
FCVT.*int.fmt*	S	W[U]/L[U]	src	RM	dest	OP-FP
FCVT.*fmt.int*	S	W[U]/L[U]	src	RM	dest	OP-FP

fcvt.w.s: Convert a floating-point number in floating-point register rs1 to a signed 32-bit in integer register rd.

fcvt.wu.s: Convert a floating-point number in floating-point register rs1 to an unsigned 32-bit in integer register rd.

fcvt.s.w: Converts a 32-bit signed integer, in integer register rs1 into a floating-point number in floating-point register rd.

fcvt.s.wu: Converts a 32-bit unsigned integer, in integer register rs1 into a floating-point number in floating-point register rd.

➢ Floating-point to floating-point sign-injection instructions.

31 — 27	26 — 25	24 — 20	19 — 15	14 — 12	11 — 7	6 — 0
funct5	fmt	rs2	rs1	rm	rd	opcode
5	2	5	5	3	5	7
FSGNJ	S	src2	src1	J[N]/JX	dest	OP-FP

fsgnj.s: Produce a result that takes all bits except the sign bit from rs1.

f[rd] = {f[rs2][31], f[rs1][30:0]}

fsgnjn.s: Produce a result that takes all bits except the sign bit is opposite of rs2's sign bit.

f[rd] = {~ f[rs2][31], f[rs1][30:0]}

fsgnjx.s: Produce a result that takes all bits except the sign bit is XOR of sign bit of rs1 and rs2.

f[rd] = {f[rs1][31] ^ f[rs2][31], f[rs1][30:0]}

➢ Instructions are provided to move bit patterns between the floating-point and integer registers.

31 — 27	26 — 25	24 — 20	19 — 15	14 — 12	11 — 7	6 — 0
funct5	fmt	rs2	rs1	rm	rd	opcode
5	2	5	5	3	5	7
FMV.X.W	S	0	src	000	dest	OP-FP
FMV.W.X	S	0	src	000	dest	OP-FP

fmv.x.w: Move the single-precision value in floating-point register rs1 represented in IEEE 754-2008 encoding to the 32 bits of integer register rd.

fmv.w.x: Move the value encoded from the lower 32 bits of integer register rs1 to the floating-point register rd.

➢ Single-Precision Floating-Point Computational Instructions:

Floating-point arithmetic instructions with one or two source operands use the R-type format.

31 27	26 25	24 20	19 15	14 12	11 7	6 0
funct5	fmt	rs2	rs1	rm	rd	opcode
5	2	5	5	3	5	7
FADD/FSUB	S	src2	src1	RM	dest	OP-FP
FMUL	S	src2	src1	RM	dest	OP-FP
FMIN-MAX	S	src2	src1	MIN/MAX	dest	OP-FP

fadd.s: Perform single-precision floating-point addition.

fsub.s: Perform single-precision floating-point subtraction.

fmul.s: Perform single-precision floating-point multiplication.

fmin.s: Write the smaller of single precision data in rs1 and rs2 to rd.

fmax.s: Write the larger of single precision data in rs1 and rs2 to rd.

➢ Single-Precision Floating-Point Compare Instructions:

Performs a quiet comparison between floating-point registers rs1, rs2 and record the Boolean result in integer register rd. Only signaling Nan inputs cause an Invalid Operation exception. The result is 0 if either operand is Nan.

31 27	26 25	24 20	19 15	14 12	11 7	6 0
funct5	fmt	rs2	rs1	rm	rd	opcode
5	2	5	5	3	5	7
FCMP	S	src2	src1	EQ/LT/LE	dest	OP-FP

feq.s: Performs a quiet equal.

flt.s: Performs a quiet less comparison.

fle.s: Performs a quiet less or equal comparison.

➢ Single-Precision Floating-Point Classify Instruction:

31 27	26 25	24 20	19 15	14 12	11 7	6 0
funct5	fmt	rs2	rs1	rm	rd	opcode
5	2	5	5	3	5	7
FCLASS	S	0	src	001	dest	OP-FP

fclass.s: Examines the value in floating-point register rs1 and writes to integer register rd a 10-bit mask that indicates the class of the floating-point number.

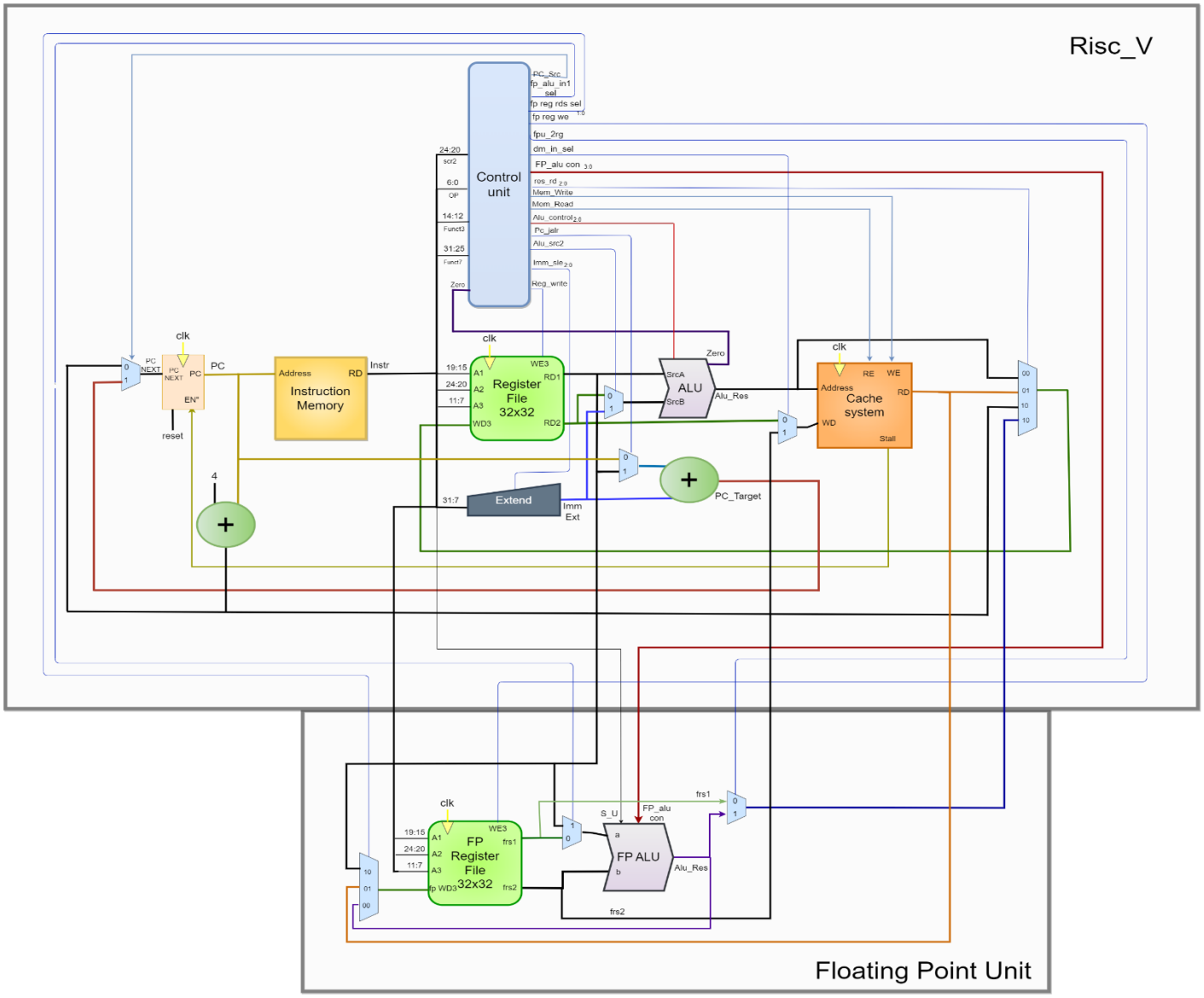
The corresponding bit in rd will be set if the property is true and clear otherwise and all other bits in rd are cleared.

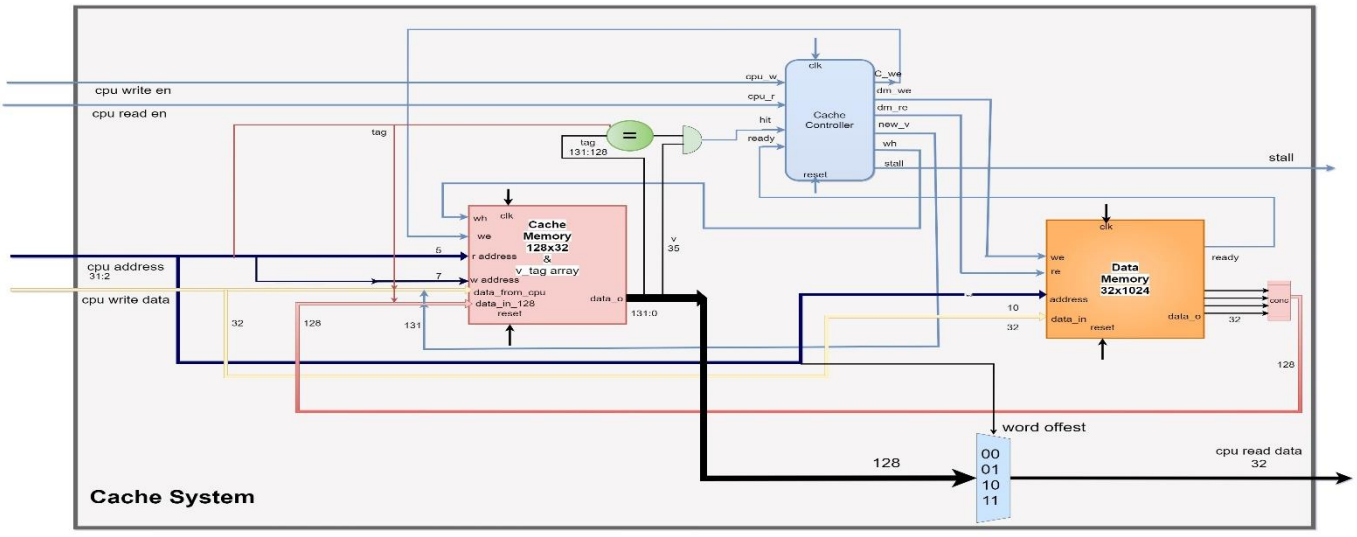
Note that exactly one bit in rd will be set.

***rd* bit**	**Meaning**
0	*rs1* is $-\infty$.
1	*rs1* is a negative normal number.
2	*rs1* is a negative subnormal number.
3	*rs1* is -0.
4	*rs1* is $+0$.
5	*rs1* is a positive subnormal number.
6	*rs1* is a positive normal number.
7	*rs1* is $+\infty$.
8	*rs1* is a signaling NaN.
9	*rs1* is a quiet NaN.

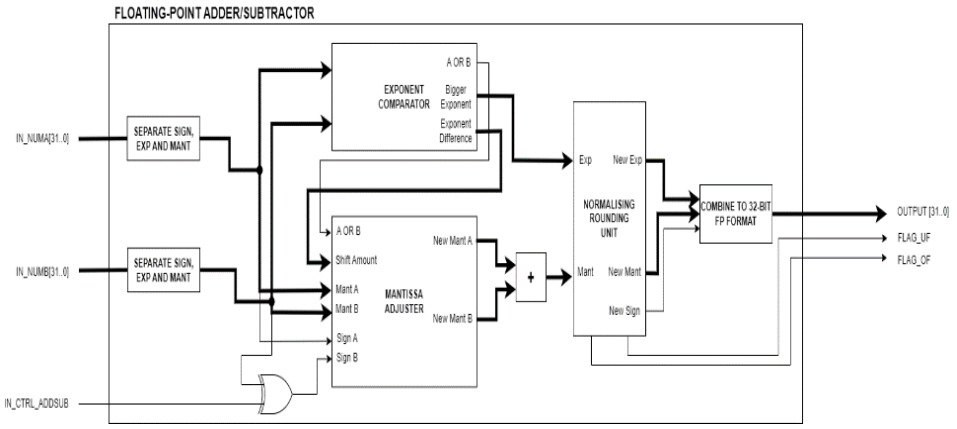
2. Architecture and RTL Block Diagram:

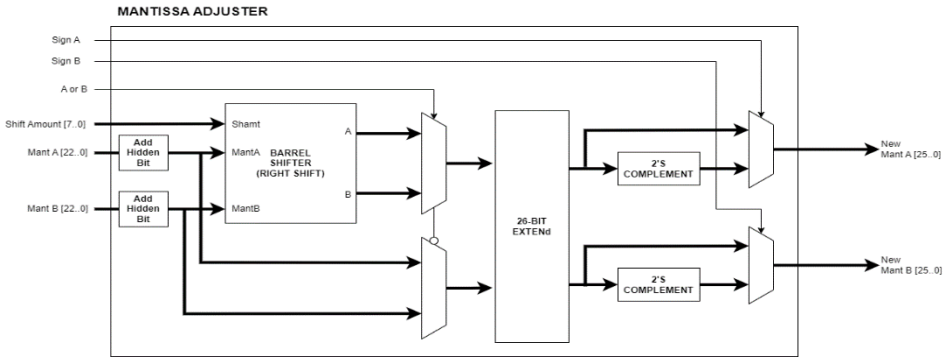
❖ RISC-V RVF Top level Block Diagram:

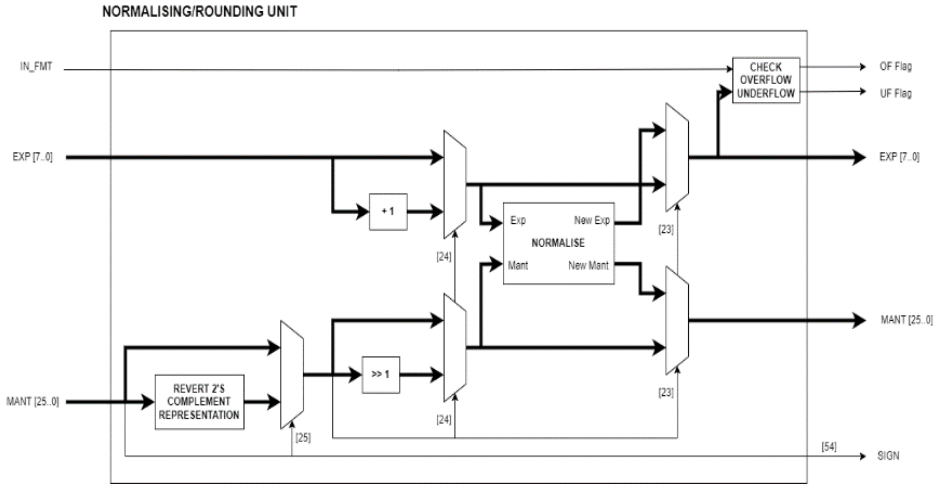


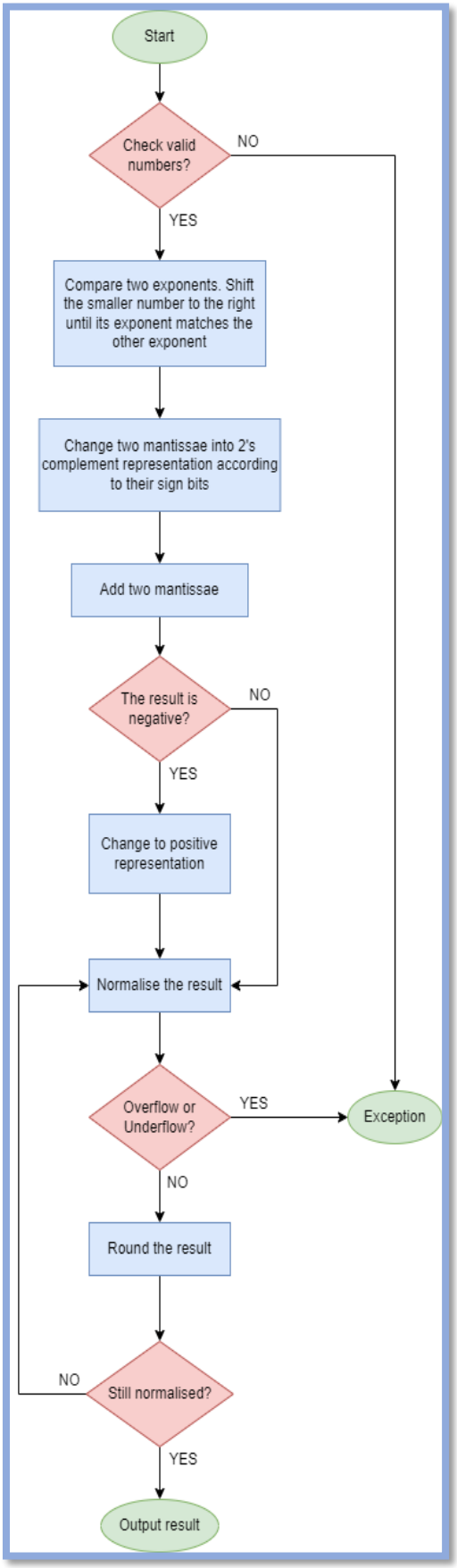


❖ Adder/Subtractor Block Diagram:

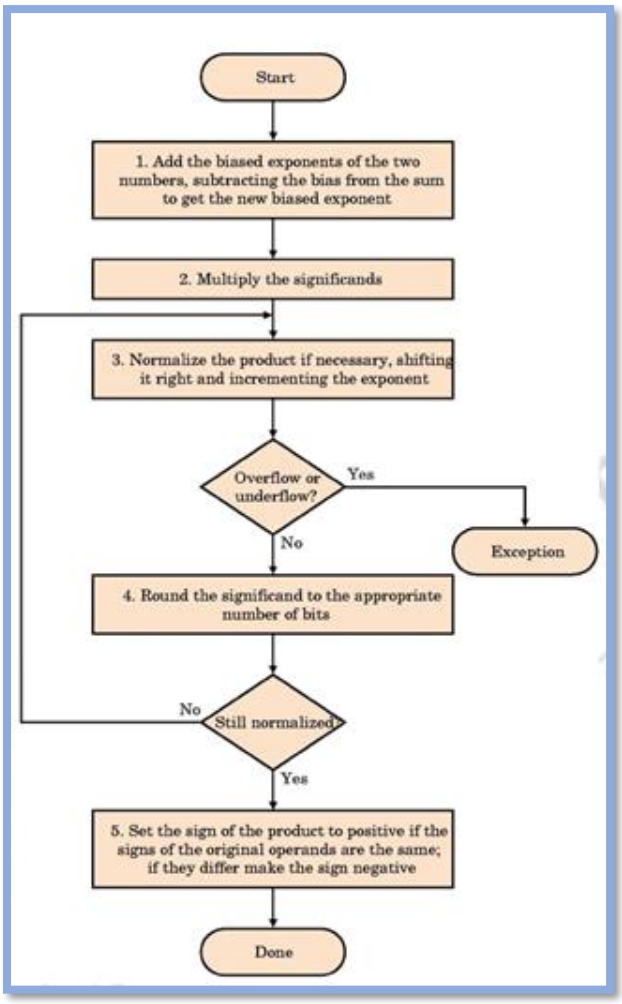


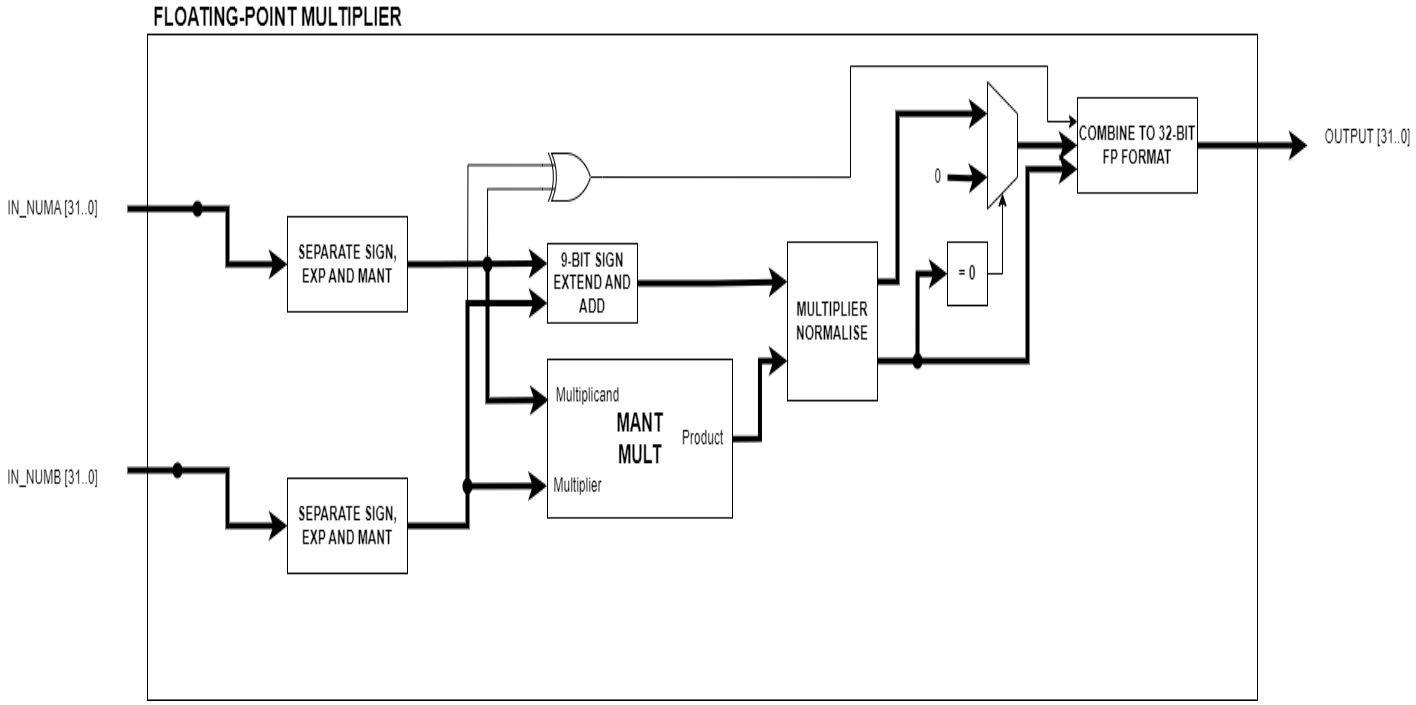




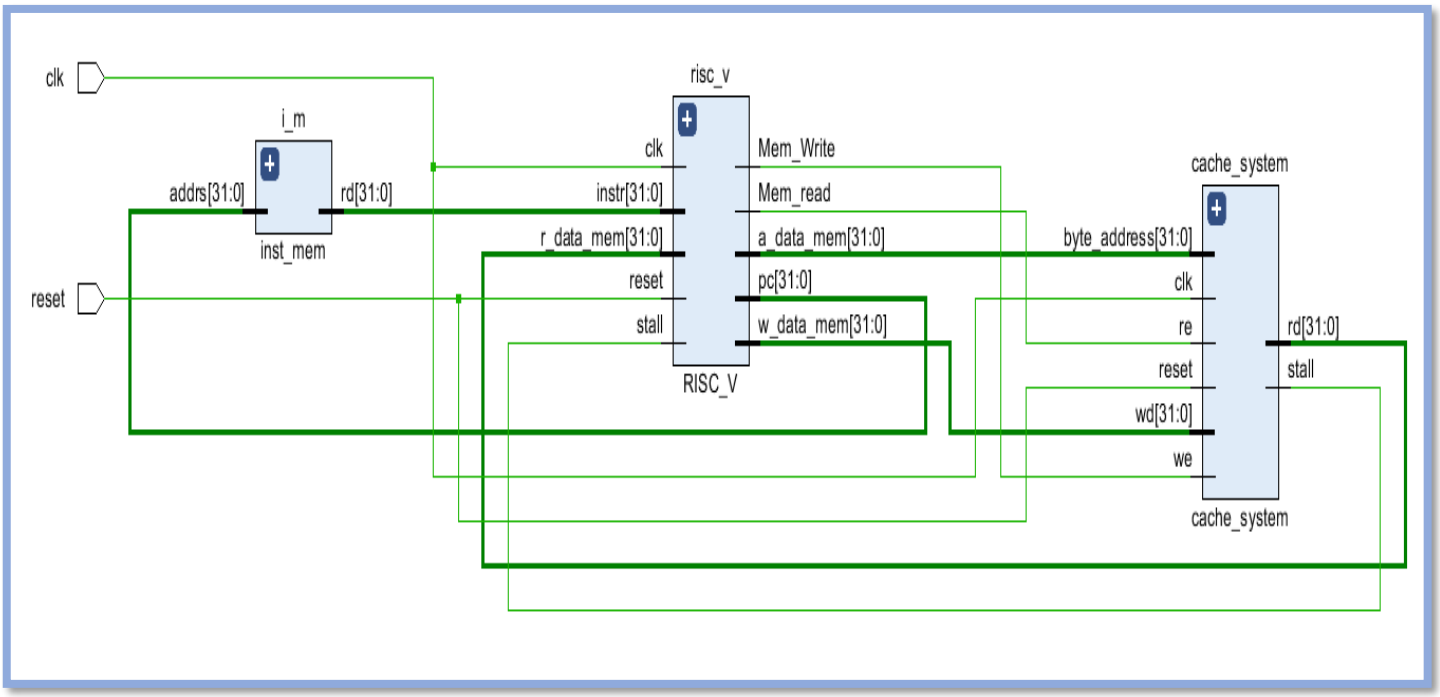


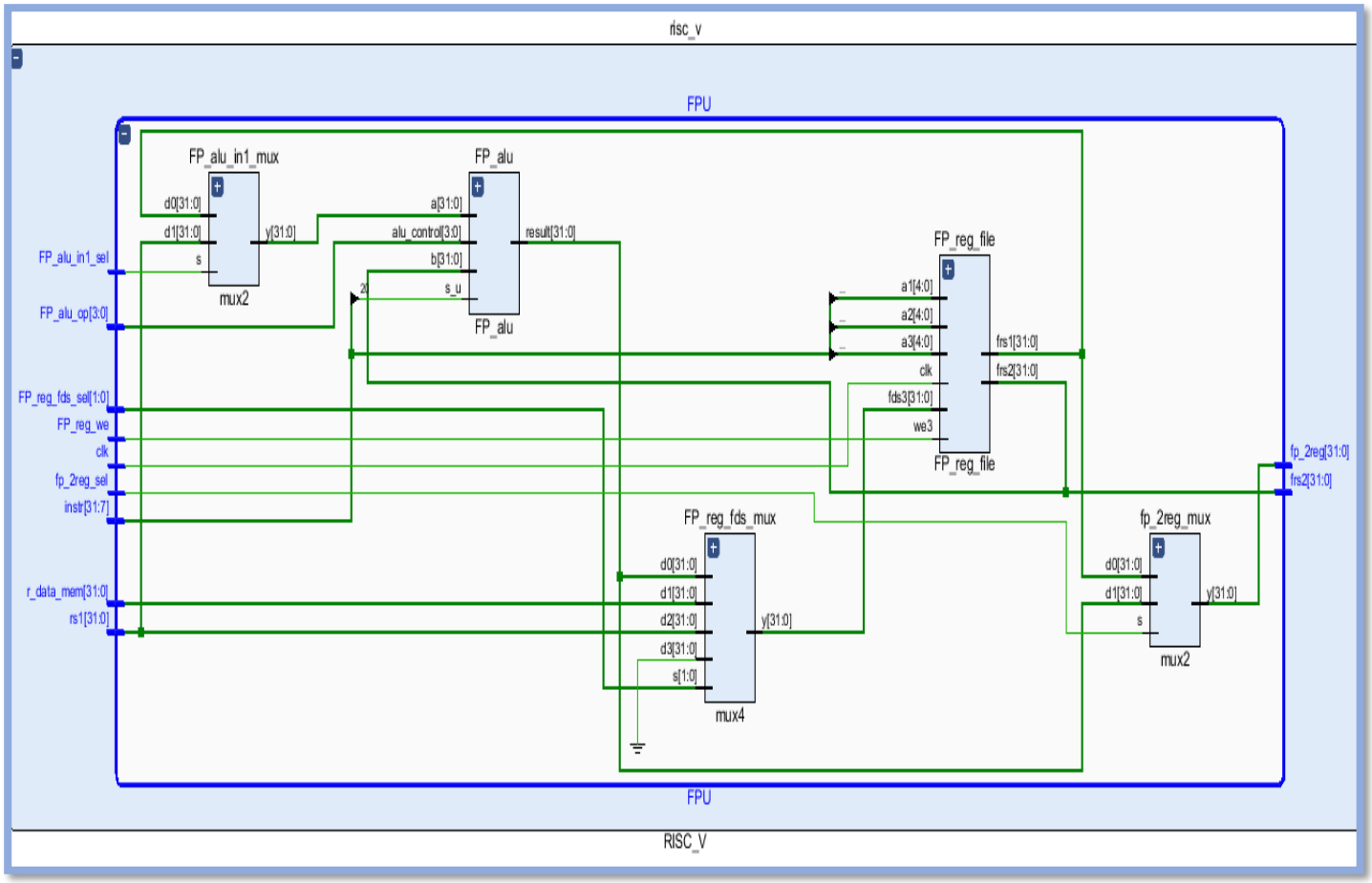
❖ Multiplier Block Diagram:

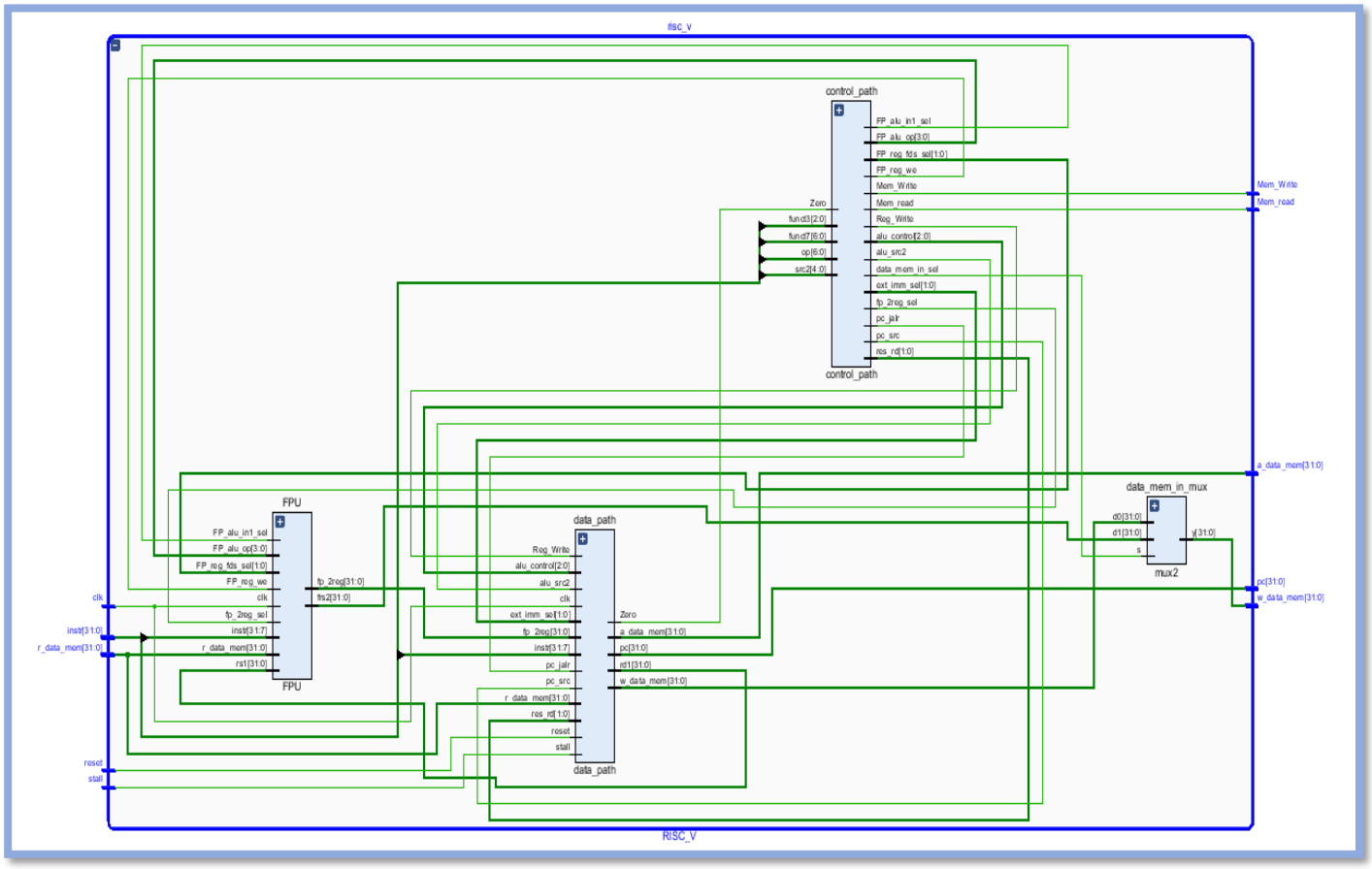


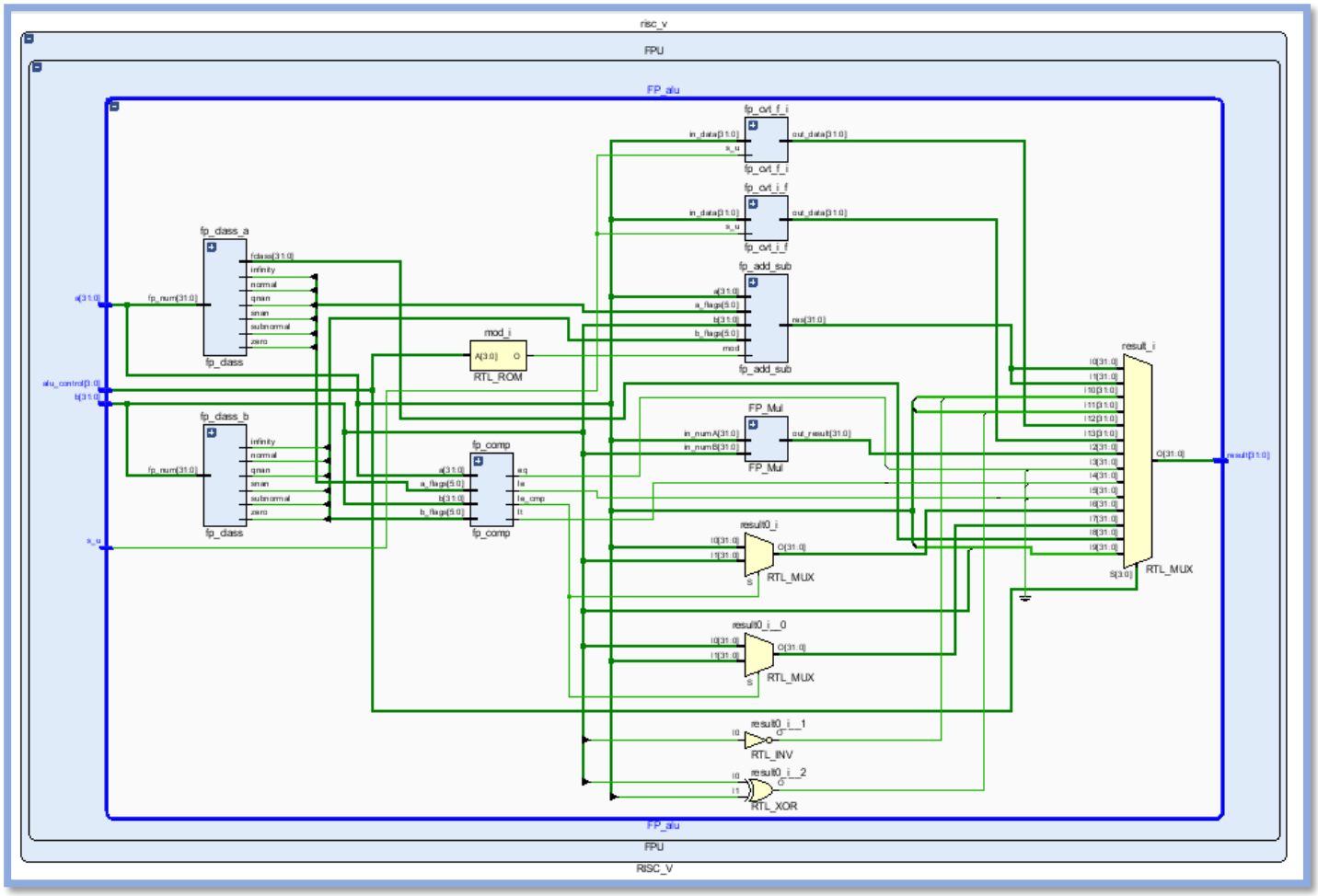


❖ RTL Block Diagram:






risc_v
control_path
FP_alu_in1_sel
FP_alu_op[3:0]
FP_reg_fds_sel[1:0]
FP_reg_we
Mem_Write
Mem_read
Reg_Write
alu_control[2:0]
alu_src2
data_mem_in_sel
ext_imm_sel[1:0]
fp_2reg_sel
pc_jalr
pc_src
res_rd[1:0]
Zero
funct3[2:0]
funct7[6:0]
op[6:0]
src[24:0]
FPU
fp_2reg[31:0]
frs2[31:0]
data_path
Zero
a_data_mem[31:0]
pc[31:0]
rd1[31:0]
w_data_mem[31:0]
r_data_mem[31:0]
res_rd[1:0]
reset
stall
data_mem_in_mux
d0[31:0]
d1[31:0]
s
y[31:0]
mux2
clk
inst[31:0]
r_data_mem[31:0]
RISC_V

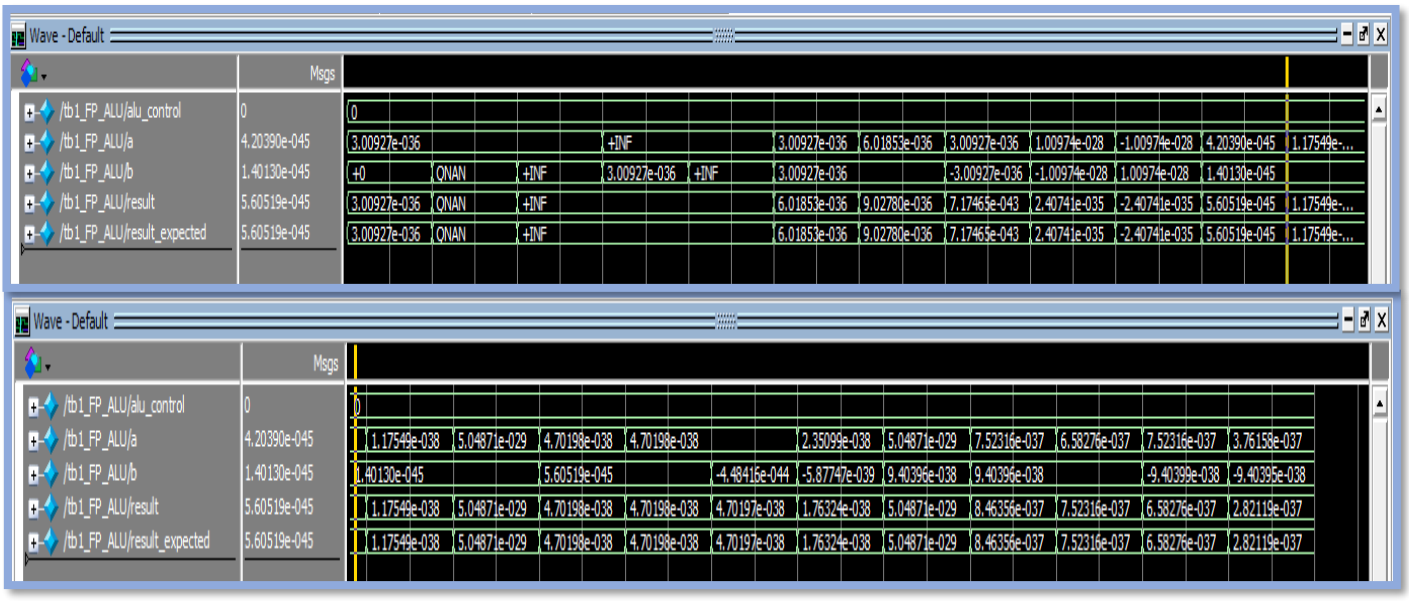

risc_v
FPU
FP_alu
fp_cvt_f_i
fp_cvt_i_f
fp_add_sub
fp_class_a
fp_class_b
fp_class
fp_comp
mod_i
RTL_ROM
FP_Mul
result0_i
result0_i__0
result0_i__1
result0_i__2
RTL_MUX
RTL_INV
RTL_XOR
result_i
RISC_V

3. Test-bench Results:

❖ FP-ALU testbench result 1.

coverage the most arithmetic cases.

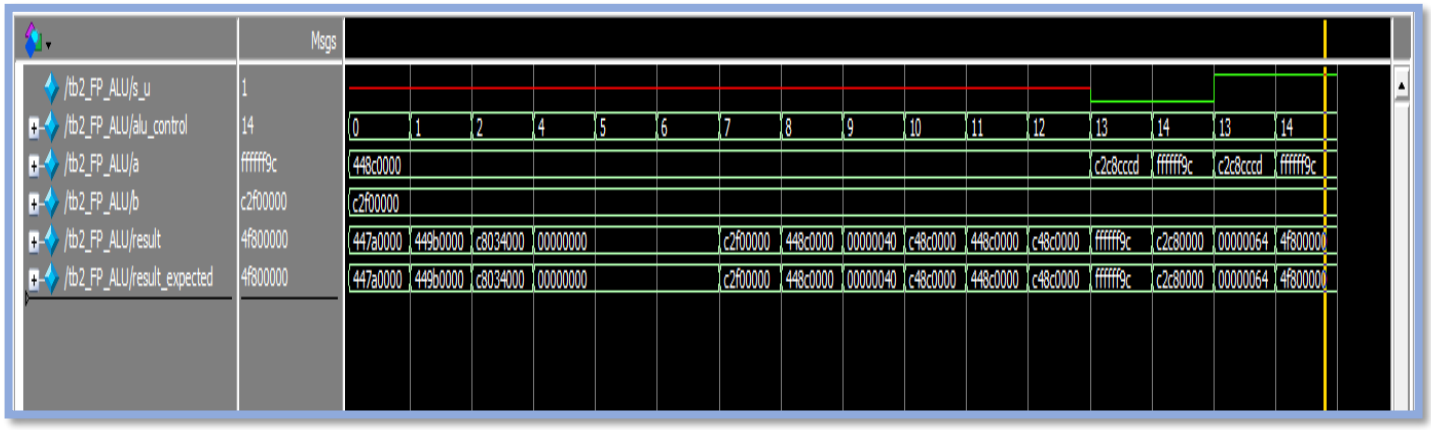
```
# Loading work.tb1_FP_ALU
VSIM 46> run -all
# Succeded case1:the test a + (+/- zero) succeded
# Succeded case2:the test a + NAN succeded
# Succeded case3:the test a + (+/- inifity) succeded
# Succeded case4:the test b + (+/- inifity) succeded
# Succeded case5:the test (+/- inifity) + (+/- inifity) succeded
# Succeded case6:the test a_expo = b_expo !=0 --> (+-1.m) + (+-1.m) --> no overflow succeded
# Succeded case7:the test a_expo = b_expo !=0 --> (+-1.m) + (+-1.m) --> overflow succeded
# Succeded case8:the test a_expo = b_expo !=0 --> (+-1.m) + (-+1.m) -->  (+>-) and test n/malize --> expo < shift succeded
# Succeded case9:the test a_expo = b_expo !=0 --> (+-1.m) + (-+1.m) -->  (+>-) and test n/malize --> expo > shift succeded
# Succeded case10:the test a_expo = b_expo !=0 --> (+-1.m) + (-+1.m) -->  (->+) and test n/malize --> expo > shift succeded
# Succeded case11:the test a_expo = b_expo = 0 --> (+-0.m) + (+-0.m) no overflow succeded
# Succeded case12:the test a_expo = b_expo = 0 --> (+-0.m) + (+-0.m) overflow succeded
# Succeded case13:the test [a_expo > b_expo] & [b_expo= 0] & [a_expo >= 23] succeded
# Succeded case14:the test [a_expo > b_expo] & [b_expo= 0] & [a_expo < 23] --> (+-1.m) + (+-0.m) overflow succeded
# Succeded case15:the test [a_expo > b_expo] & [b_expo= 0] & [a_expo < 23] --> (+-1.m) + (+-0.m) no overflow succeded
# Succeded case16:the test [a_expo > b_expo] & [b_expo= 0] & [a_expo < 23] --> (+-1.m) + (+-0.m) no b/row succeded
# Succeded case17:the test [a_expo > b_expo] & [b_expo= 0] & [a_expo < 23] --> (+-1.m) + (+-0.m) b/row succeded
# Succeded case18:the test [a_expo > b_expo] & [b_exp!o= 0] & [a_expo-b_expo >= 24] succeded
# Succeded case19:the test [a_expo > b_expo] & [b_expo!= 0] & [a_expo-b_expo < 24] --> (+-1.m) + (+-0.m) overflow succeded
# Succeded case20:the test [a_expo > b_expo] & [b_expo!= 0] & [a_expo-b_expo < 24] --> (+-1.m) + (+-0.m) no overflow succeded
# Succeded case21:the test [a_expo > b_expo] & [b_expo!= 0] & [a_expo-b_expo < 24] --> (+-1.m) + (+-0.m) no b/row succeded
# Succeded case22:the test [a_expo > b_expo] & [b_expo!= 0] & [a_expo-b_expo < 24] --> (+-1.m) + (+-0.m) b/row succeded
# ** Note: $stop    : C:/Users/moham/Music/RISC-V_FP_Extension/tb1_FP_ALU.v(244)
```



❖ FP-ALU testbench result 2.

Test all FP-ALU supported operations.

```
VSIM 180> run -all
#  Case:        Operation    In1          In2          Result      Expected_Result    Test State:              Description:
#  [Case:0  ]       Add   0x448c0000   0xc2f00000   =0x447a0000    [0x447a0000]      [Test Passed]:                          (1120.0 + -120.0=1000.0)
#  [Case:1  ]       Sub   0x448c0000   0xc2f00000   =0x449b0000    [0x449b0000]      [Test Passed]:                          (1120.0 - -120.0=1240.0)
#  [Case:2  ]       Mul   0x448c0000   0xc2f00000   =0xc8034000    [0xc8034000]      [Test Passed]:                       (1120.0 x -120.0=-134400.0)
#  [Case:4  ]       Feq   0x448c0000   0xc2f00000   =0x00000000    [0x00000000]      [Test Passed]:                             (1120.0 == -120.0)
#  [Case:5  ]       Flt   0x448c0000   0xc2f00000   =0x00000000    [0x00000000]      [Test Passed]:                              (1120.0 < -120.0)
#  [Case:6  ]       Fle   0x448c0000   0xc2f00000   =0x00000000    [0x00000000]      [Test Passed]:                             (1120.0 <= -120.0)
#  [Case:7  ]      Fmin   0x448c0000   0xc2f00000   =0xc2f00000    [0xc2f00000]      [Test Passed]:                          Min(1120.0 , -120.0)
#  [Case:8  ]      Fmax   0x448c0000   0xc2f00000   =0x448c0000    [0x448c0000]      [Test Passed]:                          Max(1120.0 , -120.0)
#  [Case:9  ]    Fclass   0x448c0000   0xc2f00000   =0x00000040    [0x00000040]      [Test Passed]:                Fclass(1120.0)= res[6]=1: +ve normal
#  [Case:10 ]     Fsgnj   0x448c0000   0xc2f00000   =0xc48c0000    [0xc48c0000]      [Test Passed]:                                          (-1120.0)
#  [Case:11 ]    Fsgnjn   0x448c0000   0xc2f00000   =0x448c0000    [0x448c0000]      [Test Passed]:                                           (1120.0)
#  [Case:12 ]    Fsgnjx   0x448c0000   0xc2f00000   =0xc48c0000    [0xc48c0000]      [Test Passed]:                                          (-1120.0)
#  [Case:13.1]  Fcvt.w.s   0xc2c8cccd   0xc2f00000   =0xffffff9c    [0xffffff9c]      [Test Passed]:                   (Cvt folat(-100.4) to S_integer(-100))
#  [Case:14.1]  Fcvt.s,w   0xffffff9c   0xc2f00000   =0xc2c80000    [0xc2c80000]      [Test Passed]:                   (Cvt S_integer(-100) to float(-100.0))
#  [Case:13.2] Fcvt.wu.s   0xc2c8cccd   0xc2f00000   =0x00000064    [0x00000064]      [Test Passed]:                   (Cvt folat(-100.4) to U_integer(100))
#  [Case:14.2] Fcvt.s,wu   0xffffff9c   0xc2f00000   =0x4f800000    [0x4f800000]      [Test Passed]:   (Cvt U_integer(-100 => (4294967196)) to float(4.2949673E9))
# ** Note: $stop    : C:/Users/moham/Music/RISC-V_FP_Extension/tb2_FP_ALU.v(174)
```



❖ RISC-V_RVF32 testbench result.

Test entire RISC-V_RVF32 with cache memory system.

Assembly Code Instructions:

RISC-V Assembly

```
1  addi x1,x0,1120
2  addi x2,x0,-120
3  fcvt.s.w f1,x1
4  fcvt.s.w f2,x2
5  fadd.s f3,f1,f2
6  fsub.s f4,f1,f2
7  fmul.s f5,f1,f2
8  feq.s x6,f1,f2
9  flt.s x7,f1,f2
10 fle.s x8,f1,f2
11 fmin.s f9,f1,f2
12 fmax.s f10,f1,f2
13 fclass.s x11,f1
14 fsgnj.s f12,f1,f2
15 fsgnjn.s f13,f1,f2
16 fsgnjx.s f14,f1,f2
17 fcvt.w.s x15,f1
18 fmv.w.x f16,x15
19 fmv.x.w x16.f16
20 fsw f14,4(x0)
21 flw f15,4(x0)
22 end:beq x0,x0,end
```

PC	Machine Code	Assembly Code	Instruction Result
0x0	0x46000093	addi x1,x0,1120	x1=0x00000460
0x4	0xF8800113	addi x2,x0,-120	x2=0xFFFFFF88
0x8	0xD00080D3	fcvt.s.w f1,x1	f1=0x448C0000
0xc	0xD0010153	fcvt.s.w f2,x2	f2=0xC2F00000
0x10	0x002081D3	fadd.s f3,f1,f2	f3=0x447a0000
0x14	0x08208253	fsub.s f4,f1,f2	f4=0x449b0000
0x18	0x102082D3	fmul.s f5,f1,f2	f5=0xc8034000
0x1c	0xA020A353	feq.s x6,f1,f2	x6=0x00000000
0x20	0xA02093D3	flt.s x7,f1,f2	x7=0x00000000
0x24	0xA0208453	fle.s x8,f1,f2	x8=0x00000000
0x28	0x282084D3	fmin.s f9,f1,f2	f9=0xC2F00000
0x2c	0x28209553	fmax.s f10,f1,f2	f10=0x448C0000
0x30	0xE00095D3	fclass.s x11,f1	x11=0x00000040
0x34	0x20208653	fsgnj.s f12,f1,f2	f12=0xc48c0000
0x38	0x202096D3	fsgnjn.s f13,f1,f2	f13=0x448c0000
0x3c	0x2020A753	fsgnjx.s f14,f1,f2	f14=0xc48c0000
0x40	0xC00087D3	fcvt.w.s x15,f1	x15=0x00000460
0x44	0xF0078853	fmv.w.x f16,x15	f16=0x00000460
0x48	0xE0080853	fmv.x.w x16,f16	x16=0x00000460
0x4c	0x00E02227	fsw f14,4(x0)	mem[4]=0xc48c0000
0x50	0x00402787	flw f15,4(x0)	f15=0xc48c0000
0x54	0x00000063	end:beq x0,x0,end	infinity loop

I-Reg_File

18	xxxxxxxx
17	xxxxxxxx
16	00000460
15	00000460
14	xxxxxxxx
13	xxxxxxxx
12	xxxxxxxx
11	00000040
10	xxxxxxxx
9	xxxxxxxx
8	00000000
7	00000000
6	00000000
5	xxxxxxxx
4	xxxxxxxx
3	xxxxxxxx
2	ffffff88
1	00000460
0	xxxxxxxx

FP-Reg_File

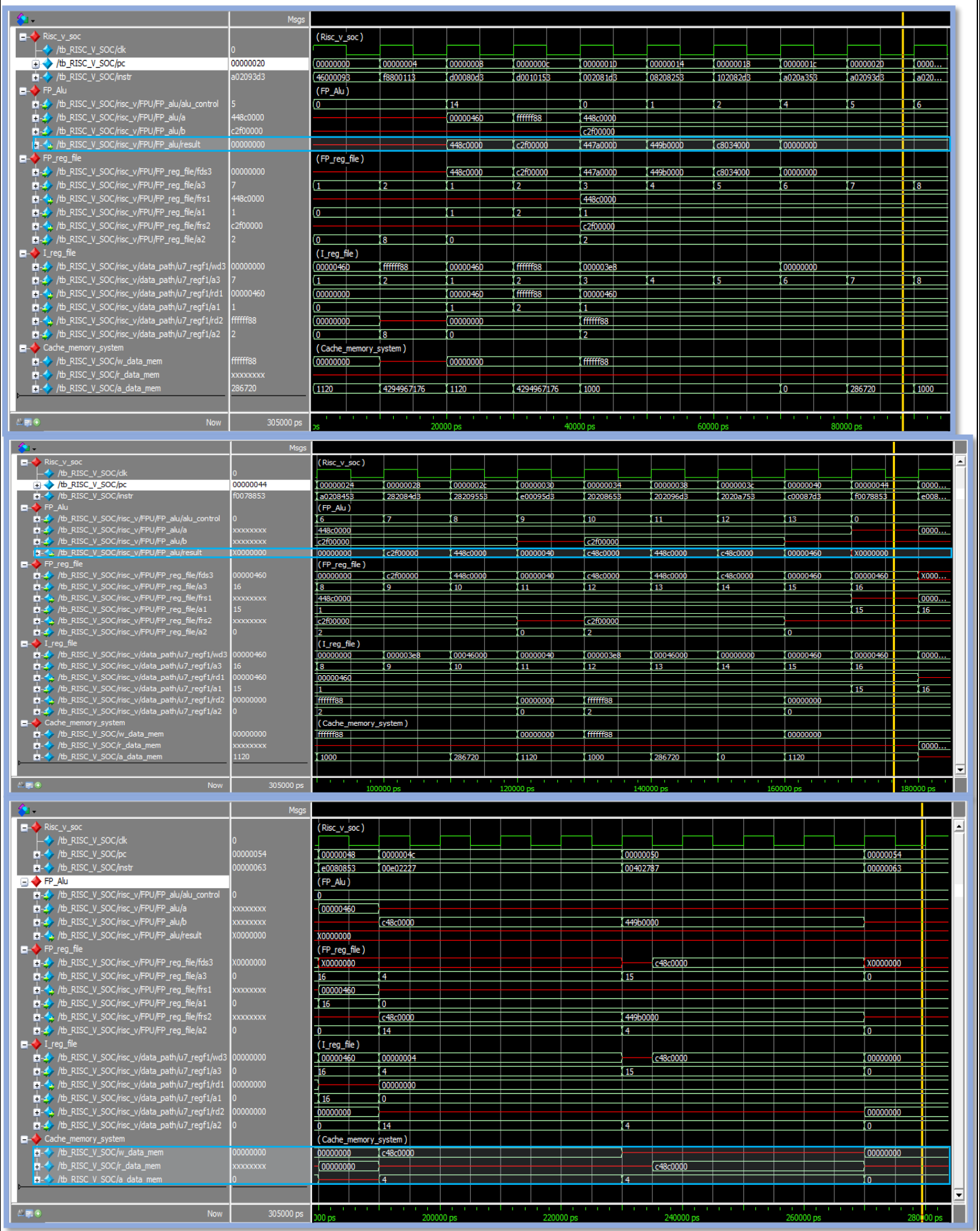
18	xxxxxxxx
17	xxxxxxxx
16	00000460
15	c48c0000
14	c48c0000
13	448c0000
12	c48c0000
11	xxxxxxxx
10	448c0000
9	c2f00000
8	xxxxxxxx
7	xxxxxxxx
6	xxxxxxxx
5	c8034000
4	449b0000
3	447a0000
2	c2f00000
1	448c0000
0	xxxxxxxx

Memory

0	xxxxxxxx
1	c48c0000
2	xxxxxxxx

Expected register files content.

Integer Reg File	Floating.P Reg File
x1=0x00000460	f1=0x448C0000
x2=0xFFFFFF88	f2=0xC2F00000
x6=0x00000000	f3=0x447a0000
x7=0x00000000	f4=0x449b0000
x8=0x00000000	f5=0xc8034000
x11=0x00000040	f9=0xC2F00000
x15=0x00000460	f10=0x448C0000
x16=0x00000460	f12=0xc48c0000
	f13=0x448c0000
	f14=0xc48c0000
	f15=0xc48c0000
	f16=0x00000460


Msgs
Risc_v_soc
/tb_RISC_V_SOC/clk
/tb_RISC_V_SOC/pc
/tb_RISC_V_SOC/instr
FP_Alu
/tb_RISC_V_SOC/risc_v/FPU/FP_alu/alu_control
/tb_RISC_V_SOC/risc_v/FPU/FP_alu/a
/tb_RISC_V_SOC/risc_v/FPU/FP_alu/b
/tb_RISC_V_SOC/risc_v/FPU/FP_alu/result
FP_reg_file
/tb_RISC_V_SOC/risc_v/FPU/FP_reg_file/fds3
/tb_RISC_V_SOC/risc_v/FPU/FP_reg_file/a3
/tb_RISC_V_SOC/risc_v/FPU/FP_reg_file/frs1
/tb_RISC_V_SOC/risc_v/FPU/FP_reg_file/a1
/tb_RISC_V_SOC/risc_v/FPU/FP_reg_file/frs2
/tb_RISC_V_SOC/risc_v/FPU/FP_reg_file/a2
I_reg_file
/tb_RISC_V_SOC/risc_v/data_path/u7_regf1/wd3
/tb_RISC_V_SOC/risc_v/data_path/u7_regf1/a3
/tb_RISC_V_SOC/risc_v/data_path/u7_regf1/rd1
/tb_RISC_V_SOC/risc_v/data_path/u7_regf1/a1
/tb_RISC_V_SOC/risc_v/data_path/u7_regf1/rd2
/tb_RISC_V_SOC/risc_v/data_path/u7_regf1/a2
Cache_memory_system
/tb_RISC_V_SOC/w_data_mem
/tb_RISC_V_SOC/r_data_mem
/tb_RISC_V_SOC/a_data_mem
Now
305000 ps
20000 ps
40000 ps
60000 ps
80000 ps
100000 ps
120000 ps
140000 ps
160000 ps
180000 ps
200000 ps
220000 ps
240000 ps
260000 ps
280000 ps

4. Design and Testbench Verilog HDL Codes:

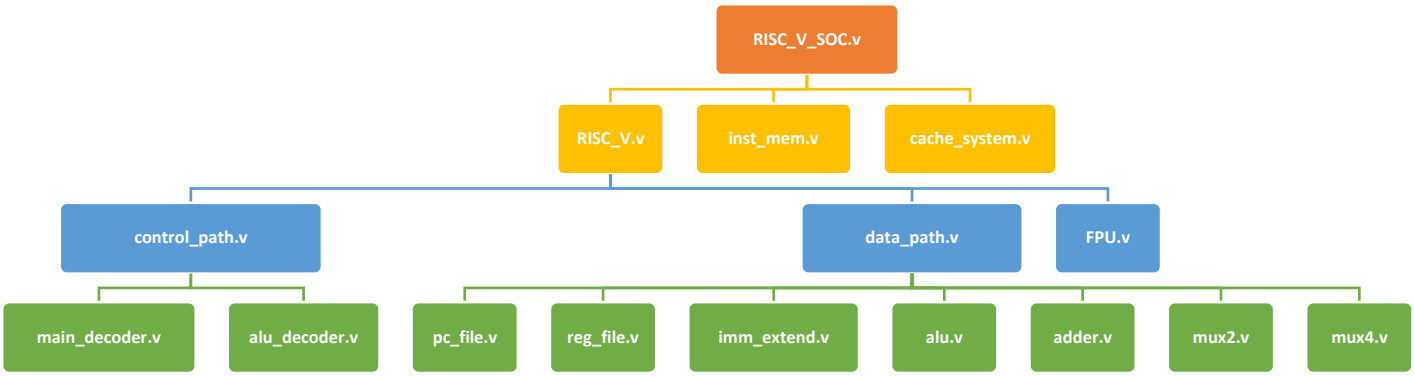
All files are in a (RISC-V_RVF32) folder.

Testbench files:

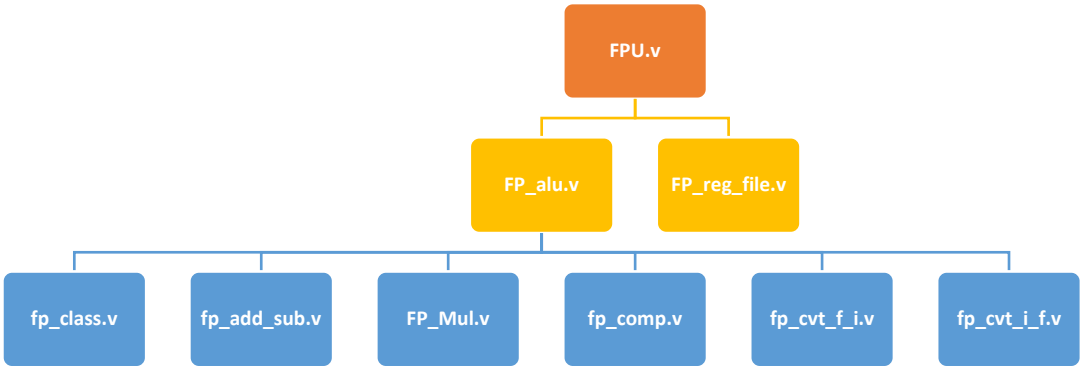
tb_RISC_V_SOC.v: //Test entire RISC-V_RVF32 with cache memory system.
tb1_FP_ALU.v: //FP_ALU testbench module_1 that coverage the most arithmetic cases.
tb2_FP_ALU.v: //FP_ALU testbench module_2 that Test all FP_Alu supported operations.

Design files:

RISC_V_SOC.v: // Top module, that integrate RISC-V_RVF32 and D_memory and I_memory.
RISC-V.v: //Top module of single cycle RISC-V_RVF32, contain control_path & data_path & Floating-point unit.



Floating-Point Unit:



Cache memory system Unit: