RISC-V Reference Guide (CREATOR Simulator)

System Calls (ecall)			
Service	**Call Code (a7)**	**Arguments**	**Result**
Print_init	1	a0 = integer	
Print_float	2	fa0 = float	
Print_double	3	fa0 = double	
Print_string	4	a0 = string addr	
Read_int	5		Integer in a0
Read_float	6		Float in fa0
Read_double	7		Double in fa0
Read_string	8	a0 = string addr a1 = length	
Sbrk	9	a0 = length	Address in a0
Exit	10		
Print_char	11	a0 = ASCII code	
Read_char	12		Char in a0

Integer Registers	
Register Name	**Usage**
zero	Constant 0
ra	Return address (routines/functions)
sp	Stack pointer
gp	Global pointer
tp	Thread pointer
t0..t6	Temporary (NOT preserved across calls)
s0..s11	Saved temporary (preserved across calls)
a0, a1	Arguments for functions / return value
a2..a7	Arguments for functions
Floating-point registers	
ft0..ft11	Temporary (NOT preserved across calls)
fs0..fs11	Saved temporary (preserved across calls)
fa0, fa1	Arguments for functions / return value
fa2..fa7	Arguments for functions

Data transfer	
li rd, n	rd = n (PseudoInst, n-> 32 bits)
mv rd, rs	rd = rs
lui rd, inm	rd = inm[31:12] <<12 (extend the sign)
Arithmetic (integer)	
add rd, rs1, rs2	rd = rs1+rs2
addi rd, rs1, n	rd = rs1 + n (n-> 12 bits)
sub rd, rs1, rs2	rd = rs1- rs2
mul rd, rs1, rs2	rd = rs1* rs2
div rd, rs1, rs2	rd = rs1/rs2
rem rd, rs1, rs2	rd = rs1% rs2
Logical (integer)	
and rd, rs1, rs2	rd = rs1 AND rs2
andi rd, rs1, n	rd = rs1 AND n (n-> 12 bits)
or rd, rs1, rs2	rd = rs1 OR rs2
ori rd, rs1, n	rd = rs1 OR n (n-> 12 bits)
not rd, rs1	rd = !rs1 (one´s complement)
neg rd, rs1	rd = !rs1 + 1 (two's complement)
xor rd, rs1, rs2	rd = rs1 XOT rs2
srli rd, rs1, n	rd = rs1 >> n logical, n-> 5 bits
slli rd, rs1, n	rd = rs1 << n n-> 5 bits
srai rd, rs1, n	rd = rs1 >> n arithmetic, n-> 5 bits
sra rd, rs1, rs2	rd = rs1 >> rs2 arithmetic
sll rd, rs1, rs2	rd = rs1 << rs2
srl rd, rs1, rs2	rd = rs1 >> rs2 logical

Branch instructions (integer registers)	
beq t0 t1 etiq	Jump to etiq if t0==t1
bne t0 t1 etiq	Jump to etiq if t0!=t1
blt t0 t1 etiq	Jump to etiq if t0<t1
bltu t0 t1 etiq	Jump to etiq if t0<t1 (unsigned)
bge t0 t1 etiq	Jump to etiq if t0>=t1
bgeu t0 t1 etiq	Jump to etiq if t0>=t1 (unsigned)
bgt t0 t1 etiq	Jump to etiq if t0>t1
bgtu t0 t1 etiq	Jump to etiq if t0>t1 (unsigned)
ble t0 t1 etiq	Jump to etiq if t0<=t1
bleu t0 t1 etiq	Jump to etiq if t0<=t1 (unsigned)
j etiq	PC = PC + etiq

Memory Access (integer registers), n→12 bits	
la rd, address	rd = address address->32 bits
lb rd, n(rs1)	rd = Memory[n+rs1] **l**oad **b**yte
lbu rd, n(rs1)	rd = Memory[n+rs1] **l**oad **b**yte **u**nsigned
lw rd, n(rs1)	rd = Memory[n+rs1] **l**oad **w**ord
sb rd, n(rs1)	Memory[n+rs1] = rd **s**tore **b**yte
sw rd, n(rs1)	Memory[n+rs1] = sw **s**tore **w**ord

Arithmetic (floating-point, .s/.d)	
fmv.s fd, fs1	fd = fs1
fadd.s fd, fs1, fs2	fd = fs1+fs2
fsub.s fd, fs1, fs2	fd = fs1-fs2
fmul.s fd, fs1, fs2	fd = fs1*fs2
fdiv.s fd, fs1, fs2	fd = fs1/fs2
fmin.s fd, fs1, fs2	fd = min(fs1,fs2)
fmax.s fd, fs1, fs2	fd = max(fs1,fs2)
fsqrt.s fd, fs1	fd = sqrt(fs1)
fmadd.s fd, fs1, fs2, fs3	fd = fs1*fs2+fs3
fmsub.s fd, ff1, fs2, fs3	fd = fs1*fs2-fs3
fabs.s fd, fs1	fd = |fs|
fneg.s fd, fs1	fd = -fs
Integer ←→ Floating point	
fmv.w.x fd, rs	fd = rs single = integer
fmv.x.w rd, fs	rd = fs integer = single
Comparison (integer), n→ 12 bits	
slt rd, rs1, rs2	if (s(rs1) < s(rs2)) rd = 1; else rd = 0
sltu rd, rs1, rs2	if (u(rs1) < u(rs2)) rd = 1; else rd = 0
slti rd, rs1, n	if (s(rs1) < s(n)) rd = 1; else rd = 0
sltiu rd, rs1, n	if (u(rs1) < u(5)) rd = 1; else rd = 0
seqz rd, rs1	if (rs1 == 0) rd = 1; else rd = 0
snez rd, rs1	if (rs1 != 0) rd = 1; else rd = 0
sgtz rd, rs1	if (rs1 > 0) rd = 1; else rd = 0
sltz rd, rs1	if (rs1 < 0) rd = 1; else rd = 0

Comparison (floating point) (rd=int register, fs1 and fs2 floating point register)	
feq.s rd, fs1, fs2	if (fs1== fs2) rd= 1;else rd = 0 (float)
fle.s rd, fs1, fs2	if (fs1<= fs2) rd= 1;else rd = 0 (float)
flt.s rd, fs1, fs2	if (fs1< fs2) rd= 1;else rd = 0 (float)
feq.d rd, fs1, fs2	if (fs1== fs2) rd= 1;else rd = 0 (double)
fle.d rd, fs1, fs2	if (fs1<= fs2) rd= 1;else rd = 0 (double)
flt.d rd, fs1, fs2	if (fs1< fs2) rd= 1;else rd = 0 (double)
Function Calls	
jal ra, address	ra = PC; PC = address
jr ra	PC = ra
Hardware Counter	
rdcycle rd	rd = number of elapsed cycles

Memory access (floating point), n→12bits	
flw fd, n(rs1)	fd = Memory[n+rs1] **l**oad **f**loat
fsw fd, n(rs1)	Memory[n+rs1] = fd **s**tore **f**loat
fld fd, n(rs1)	fd = Memory[n+rs1] **l**oad **d**ouble
fsd fd, n(rs1)	Memory[n+rs1] = fd **s**tore **d**ouble

Conversion Operations	
fcvt.w.s rd, fs1	From single precision (fs1) to integer (rd) with sign
fcvt.wu.s rd, fs1	From single precision (fs1) to integer (rd) without sign
fcvt.s.w fd, rs1	From integer with sign (rs1) to single precision (fd)
fcvt.s.wu fd, rs1	From integer without sign (rs1) to single precision (fd)
fcvt.w.d rd, fs1	From rom double precision (fs1) to integer (rd) with sign
fcvt.wu.d rd, fs1	From double precision (fs1) to integer (rd) without sign
fcvt.d.w fd, rs1	From integer with sign (rs1) to double precision (fd)
fcvt.d.wu fd, rs1	From integer without sign (rs1) to double precision (fd)
fcvt.s.d fd, fs1	From double (fs1) to single precision (fd)
fcvt.d.s fd, fs1	From single (fs1) to double precision (fd)

Floating-pont Clasification	
fclass.s rd, fs1	Classify single precision
fclass.d rd, fs1	Classify double precision
Value in rd	**Meaning**
0, 7	-Inf, +Inf
1	Normalized negative
2	Not normalized negative
3, 4	-0, +0
5	Normalized positive
6	Not normalized positive
8, 9	NaN

CC BY NC SA