RISC-V: guía de referencia (CREATOR)

Llamada al sistema (ecall)			
Servicio	**Código**	**Argumentos**	**Resultados**
print_init	1	a0 = entero	
print_float	2	fa0 = float	
print_double	3	fa0 = double	
print_string	4	a0 = dir. string	
read_int	5		entero en a0
read_float	6		float en fa0
read_double	7		double in fa0
read_string	8	a0 = dir. string a1 = longitud	
sbrk	9	a0 = # bytes	dir. en a0
exit	10		
print_char	11	a0 = código ASCII	
read_char	12		char en a0

Registro de enteros	
Nombre registro	**Uso**
zero	Constante 0
ra	Dirección retorno (rutinas/funciones)
sp	Puntero de pila
gp	Puntero global
tp	Puntero de hilo
t0..t6	Temporal (NO se preserva en llamadas)
s0..s11	Temp. Guardados (se preserva en llam.)
a0, a1	Arg. para funciones / valor retorno
a2..a7	Argumentos para funciones
Registros de coma flotante	
ft0..ft11	Temporal (NO se preserva en llamadas)
fs0..fs11	Temp. guardados (se preserva en llam.)
fa0, fa1	Arg. para funciones / valor retorno
fa2..fa7	Argumentos para funciones

Transferencia de datos	
li rd, n	rd = n (PseudoInst, n-> 32 bits)
mv rd, rs	rd = rs
lui rd, inm	rd = inm[31:12] <<12 (extensión signo)
Aritméticas (enteros)	
add rd, rs1, rs2	rd = rs1 + rs2
addi rd, rs1, n	rd = rs1 + n
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
Lógicas (entero)	
and rd, rs1, rs2	rd = rs1 AND rs2
andi rd, rs, 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 (complemento a uno)
neg rd, rs1	rd = !rs1 + 1 (complemento a dos)
xor rd, rs1, rs2	rd = rs1 XOT rs2
srli rd, rs1, n	rd = rs1 >> n (derecha lógico)
slli rd, rs1, n	rd = rs1 << n (n-> 5 bits)
srai rd, rs1, n	rd = rs1 >> n (derecha aritmético)
sra rd, rs1, rs2	rd = rs1 >> rs2 (derecha aritmético)
sll rd, rs1, rs2	rd = rs1 << rs2
srl rd, rs1, rs2	rd = rs1 >> rs2 (derecha lógico)

Instrucciones de salto (registros de enteros)	
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

Acceso a memoria (registro de enteros)	
la rd, address	rd = dirección dirección->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 unsigned
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] = sd **s**tore **w**ord

Operaciones de conversión	
fcvt.w.s rd, rs1	De simple precisión (fs1) a entero (rd) con signo
fcvt.wu.s rd, rs1	De simple precisión (fs1) a entero (rd) sin signo
fcvt.s.w rd, rs1	De entero con signo (rs1) a simple precisión (rd)
fcvt.s.wu rd, rs1	De entero sin signo (rs1) a simple precisión (rd)
fcvt.w.d rd, rs1	De doble precisión (fs1) a entero (rd) con signo
fcvt.wu.d rd, rs1	De doble precisión (fs1) a entero (rd) sin signo
fcvt.d.w rd, rs1	De entero con signo (rs1) a double precisión (rd)
fcvt.d.wu rd, rs1	De entero sin signo (rs1) a double precisión (rd)
fcvt.s.d rd, rs1	De doble precisión (rs1) a simple precisión (rd)
fcvt.d.s rd, rs1	De simple precisión (rs1) a double precisión(rd)

Aritméticas (coma flotante, .s/.d)	
fmv.s	rd = rs
fadd.s rd, rs1, rs2	rd = rs1 + rs2
fsub.s rd, rs1, rs2	rd = rs1 - rs2
fmul.s rd, rs1, rs2	rd = rs1 * rs2
fdiv.s rd, rs1, rs2	rd = rs1 / rs2
fmin.s rd, rs1, rs2	rd = min(rs1,rs2)
fmax.s rd, rs1, rs2	rd = max(rs1,rs2)
fsqrt.s rd, rs	rd = sqrt(rs)
fmadd.s rd, rs1, rs2, rs3	rd = rs1 * rs2 + rs3
fmsub.s rd, rs1, rs2, rs3	rd = rs1 * rs2 - rs3
fabs.s rd, rs	rd = |rs|
fneg.s rd, rs	rd = -rs
Entero ←→ Coma flotante	
fmv.w.x rd, rs	rd = rs simple = entero
fmv.x.w rd, rs	rd = rs entero = simple
Comparación (enteros), 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

Comparación (coma flotante) (rd=reg. entero, rs1 y rs2 reg. de coma flotante)	
feq.s rd, rs1, rs2	if (rs1== rs2) rd= 1;else rd = 0 (float)
fle.s rd, rs1, rs2	if (rs1<= rs2) rd= 1;else rd = 0 (float)
flt.s rd, rs1, rs2	if (rs1< rs2) rd= 1;else rd = 0 (float)
feq.d rd, rs1, rs2	if (rs1== rs2) rd= 1;else rd = 0 (double)
fle.d rd, rs1, rs2	if (rs1<= rs2) rd= 1;else rd = 0 (double)
flt.d rd, rs1, rs2	if (rs1< rs2) rd= 1;else rd = 0 (double)
Llamadas a función	
jal ra, address	ra = PC; PC = address
jr ra	PC = ra
Hardware Counter	
rdcycle rd	rd = número de ciclos de reloj usados

Acceso a memoria (coma flotante)	
flw rd, n(rs1)	rd = Memoria[n+rs1] **l**oad **f**loat
fsw rd, n(rs1)	Memoria[n+rs1] = rd **s**tore **f**loat
fld rd, n(rs1)	rd = Memoria[n+rs1] **l**oad **d**ouble
fsd rd, n(rs1)	Memoria[n+rs1] = rd **s**tore **d**ouble

Clasificación de coma flotante	
fclass.s rd, rs1	Clasifica simple precisión
fclass.d rd, rs1	Clasifica doble precisión
Valor en rd	**Significado**
0, 7	-Inf, +Inf
1	Normalizado negativo
2	No normalizado negative
3, 4	-0, +0
5	Normalizado positivo
6	Not normalizado positivo
8, 9	NaN