Referencia para el parcial

Set de instrucciones RISC-V

Formatos de instrucción

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

RV32I Instrucciones base de enteros

Inst	Nombre	FMT	Opcode	funct3	funct7	Descripción (C)	Nota
`add`	ADD	R	`0110011`	`0x0`	`0x00`	`rd = rs1 + rs2`	
`sub`	SUB	R	`0110011`	`0x0`	`0x20`	`rd = rs1 - rs2`	
`xor`	XOR	R	`0110011`	`0x4`	`0x00`	`rd = rs1 ^ rs2`	
`or`	OR	R	`0110011`	`0x6`	`0x00`	`rd = rs1 | rs2`	
`and`	AND	R	`0110011`	`0x7`	`0x00`	`rd = rs1 & rs2`	
`sll`	Desplaz. lógico izq.	R	`0110011`	`0x1`	`0x00`	`rd = rs1 << rs2`	
`srl`	Desplaz. lógico der.	R	`0110011`	`0x5`	`0x00`	`rd = rs1 >> rs2`	
`sra`	Desplaz. aritmético der.	R	`0110011`	`0x5`	`0x20`	`rd = rs1 >> rs2`	extiende-signo
`slt`	Asigna si <	R	`0110011`	`0x2`	`0x00`	`rd = (rs1 < rs2)?1:0`	
`sltu`	Asigna si < (U)	R	`0110011`	`0x3`	`0x00`	`rd = (rs1 < rs2)?1:0`	extiende-ceros
`addi`	ADD Inmediato	I	`0010011`	`0x0`		`rd = rs1 + imm`	
`xori`	XOR Inmediato	I	`0010011`	`0x4`		`rd = rs1 ^ imm`	
`ori`	OR Inmediato	I	`0010011`	`0x6`		`rd = rs1 | imm`	
`andi`	AND Inmediato	I	`0010011`	`0x7`		`rd = rs1 & imm`	
`slli`	Desplaz. lóg. izq. Inm	I	`0010011`	`0x1`	`imm[5:11]=0x00`	`rd = rs1 << imm[0:4]`	
`srli`	Desplaz. lóg. der. Inm	I	`0010011`	`0x5`	`imm[5:11]=0x00`	`rd = rs1 >> imm[0:4]`	
`srai`	Desplaz. arit. der. Imm	I	`0010011`	`0x5`	`imm[5:11]=0x20`	`rd = rs1 >> imm[0:4]`	extiende-signo
`slti`	Asig. si < Imm	I	`0010011`	`0x2`		`rd = (rs1 < imm)?1:0`	
`sltiu`	Asig. si < Imm (U)	I	`0010011`	`0x3`		`rd = (rs1 < imm)?1:0`	zero-extends
`lb`	Carga Byte	I	`0000011`	`0x0`		`rd = M[rs1+imm][0:7]`	
`lh`	Carga Half	I	`0000011`	`0x1`		`rd = M[rs1+imm][0:15]`	
`lw`	Carga Word	I	`0000011`	`0x2`		`rd = M[rs1+imm][0:31]`	
`lbu`	Carga Byte (U)	I	`0000011`	`0x4`		`rd = M[rs1+imm][0:7]`	extiende-ceros
`lhu`	Carga Half (U)	I	`0000011`	`0x5`		`rd = M[rs1+imm][0:15]`	extiende-ceros
`sb`	Guarda Byte	S	`0100011`	`0x0`		`M[rs1+imm][0:7] = rs2[0:7]`	
`sh`	Guarda Half	S	`0100011`	`0x1`		`M[rs1+imm][0:15] = rs2[0:15]`	
`sw`	Guarda Word	S	`0100011`	`0x2`		`M[rs1+imm][0:31] = rs2[0:31]`	
`beq`	Salto cond. ==	B	`1100011`	`0x0`		`if(rs1 == rs2) PC += imm`	
`bne`	Salto cond. !=	B	`1100011`	`0x1`		`if(rs1 != rs2) PC += imm`	
`blt`	Salto cond. <	B	`1100011`	`0x4`		`if(rs1 <  rs2) PC += imm`	
`bge`	Salto cond. ≥	B	`1100011`	`0x5`		`if(rs1 >= rs2) PC += imm`	
`bltu`	Salto cond. < (U)	B	`1100011`	`0x6`		`if(rs1 <  rs2) PC += imm`	zero-extends
`bgeu`	Salto cond. ≥ (U)	B	`1100011`	`0x7`		`if(rs1 >= rs2) PC += imm`	zero-extends
`jal`	Jump And Link	J	`1101111`			`rd = PC+4; PC += imm`	
`jalr`	Jump And Link Reg	I	`1100111`	`0x0`		`rd = PC+4; PC = rs1 + imm`	
`lui`	Carga Inm. superior	U	`0110111`			`rd = imm << 12`	
`auipc`	Carga Inm. superior a PC	U	`0010111`			`rd = PC + (imm << 12)`	
`ecall`	Llamada de entorno	I	`1110011`	`0x0`	`imm=0x0`	`Transfer control to OS`	
`ebreak`	Parada de entorno	I	`1110011`	`0x0`	`imm=0x1`	`Transfer control to debugger`	

Pseudo instrucciones

Pseudoinstrucción	Instrucción base	Significado
`la rd, symbol`	`auipc rd, symbol[31:12]` `addi rd, rd, symbol[11:0]`	Carga dirección
`l{b|h|w|d} rd, symbol`	`auipc rd, symbol[31:12]` `l{b|h|w|d} rd, symbol[11:0](rd)`	Carga global

`s{b|h|w|d} rd, symbol, rt`	`auipc rt, symbol[31:12]`
	`s{b|h|w|d} rd, symbol[11:0](rt)`
	`fs{w|d} rd, symbol[11:0](rt)`
		Guarda global

`nop`	`addi x0, x0, 0`	Sin operación
`li rd, immediate`	*Myriad sequences*	Carga inmediato
`mv rd, rs`	`addi rd, rs, 0`	Copia registro
`not rd, rs`	`xori rd, rs, -1`	Inversión lógica
`neg rd, rs`	`sub rd, x0, rs`	Complemento a dos
`seqz rd, rs`	`sltiu rd, rs, 1`	Asigna si $=$ cero
`snez rd, rs`	`sltu rd, x0, rs`	Asigna si $\neq$ cero
`sltz rd, rs`	`slt rd, rs, x0`	Asigna si $<$ cero
`sgtz rd, rs`	`slt rd, x0, rs`	Asigna si $>$ cero
`beqz rs, offset`	`beq rs, x0, offset`	Salta si $=$ cero
`bnez rs, offset`	`bne rs, x0, offset`	Salta si $\neq$ cero
`blez rs, offset`	`bge x0, rs, offset`	Salta si $\leq$ cero
`bgez rs, offset`	`bge rs, x0, offset`	Salta si $\geq$ cero
`bltz rs, offset`	`blt rs, x0, offset`	Salta si $<$ cero
`bgtz rs, offset`	`blt x0, rs, offset`	Salta si $>$ cero
`bgt rs, rt, offset`	`blt rt, rs, offset`	Salta si $>$
`ble rs, rt, offset`	`bge rt, rs, offset`	Salta si $\leq$
`bgtu rs, rt, offset`	`bltu rt, rs, offset`	Salta si $>$,sin signo
`bleu rs, rt, offset`	`bgeu rt, rs, offset`	Salta si $\leq$,sin signo
`j offset`	`jal x0, offset`	Salto incond.
`jal offset`	`jal x1, offset`	Jump and link
`jr rs`	`jalr x0, rs, 0`	Salta a registro
`jalr rs`	`jalr x1, rs, 0`	Jump and link a reg.
`ret`	`jalr x0, x1, 0`	Vuelve de función

RV32M Extensiones de multiplicación

Inst	Nombre	FMT	Opcode	funct3	funct7	Descripción (C)
`mul`	MUL	R	`0110011`	`0x0`	`0x01`	`rd = (rs1 * rs2)[31:0]`
`div`	DIV	R	`0110011`	`0x4`	`0x01`	`rd = rs1 / rs2`
`rem`	Resto	R	`0110011`	`0x6`	`0x01`	`rd = rs1 % rs2`
`remu`	Resto (U)	R	`0110011`	`0x7`	`0x01`	`rd = rs1 % rs2`

Registros y convención

Registro	Nombre convención	Descripción	Lo guarda
`x0`	`zero`	Constante en cero	—
`x1`	`ra`	Dirección de retorno	Función llamadora
`x2`	`sp`	Stack pointer	Función llamada
`x3`	`gp`	Global pointer (*)	—
`x4`	`tp`	Thread pointer(*)	—
`x5-x7`	`t0-t2`	Registros volátiles	Función llamadora
`x8`	`s0 / fp`	Preservado / frame pointer	Función llamada
`x9`	`s1`	Registro preservado	Función llamada
`x10-x11`	`a0-a1`	Valores de retorno	Función llamadora
`x12-x17`	`a2-a7`	Registros de argumentos	Función llamadora
`x18-x27`	`s2-s11`	Registros preservados	Función llamada
`x28-x31`	`t3-t6`	Registros volátiles	Función llamadora

Las descripciones marcadas con (*) son de cosas que no vemos en la materia.