31 25	24 20	19 15	14 12	11 7	6 0	
imm[31:12]				rd	0110111	U lui
imm[31:12]				rd	0010111	U auipc
imm[20|10:1|11|19:12]				rd	1101111	J jal
imm[11:0]		rs1	000	rd	1100111	I jalr
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	B beq
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	B bne
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	B blt
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	B bge
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	B bltu
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	B bgeu
imm[11:0]		rs1	000	rd	0000011	I lb
imm[11:0]		rs1	001	rd	0000011	I lh
imm[11:0]		rs1	010	rd	0000011	I lw
imm[11:0]		rs1	100	rd	0000011	I lbu
imm[11:0]		rs1	101	rd	0000011	I lhu
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	S sb
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	S sh
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	S sw
imm[11:0]		rs1	000	rd	0010011	I addi
imm[11:0]		rs1	010	rd	0010011	I slti
imm[11:0]		rs1	011	rd	0010011	I sltiu
imm[11:0]		rs1	100	rd	0010011	I xori
imm[11:0]		rs1	110	rd	0010011	I ori
imm[11:0]		rs1	111	rd	0010011	I andi
0000000	shamt	rs1	001	rd	0010011	I slli
0000000	shamt	rs1	101	rd	0010011	I srli
0100000	shamt	rs1	101	rd	0010011	I srai
0000000	rs2	rs1	000	rd	0110011	R add
0100000	rs2	rs1	000	rd	0110011	R sub
0000000	rs2	rs1	001	rd	0110011	R sll
0000000	rs2	rs1	010	rd	0110011	R slt
0000000	rs2	rs1	011	rd	0110011	R sltu
0000000	rs2	rs1	100	rd	0110011	R xor
0000000	rs2	rs1	101	rd	0110011	R srl
0100000	rs2	rs1	101	rd	0110011	R sra
0000000	rs2	rs1	110	rd	0110011	R or
0000000	rs2	rs1	111	rd	0110011	R and
0000 pred	succ	00000	000	00000	0001111	I fence
0000 0000	0000	00000	001	00000	0001111	I fence.i
000000000000		00000	000	00000	1110011	I ecall
000000000001		00000	000	00000	1110011	I ebreak
csr		rs1	001	rd	1110011	I csrrw
csr		rs1	010	rd	1110011	I csrrs
csr		rs1	011	rd	1110011	I csrrc
csr		zimm	101	rd	1110011	I csrrwi
csr		zimm	110	rd	1110011	I csrrsi
csr		zimm	111	rd	1110011	I csrrci

Figura 2.3: El mapa de opcodes de RV32I tiene la estructura de la instrucción, opcodes, tipo de formato y nombres (La Tabla 19.2 de [Waterman and Asanović 2017] es la base de esta figura).

Registro	Nombre ABI	Descripción	¿Preservado en llamadas?
x0	zero	Alambrado a cero	—
x1	ra	Dirección de retorno	No
x2	sp	Stack pointer	Sí
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Link register temporal/alterno	No
x6–7	t1–2	Temporales	No
x8	s0/fp	Saved register/frame pointer	Sí
x9	s1	Saved register	Sí
x10–11	a0–1	Argumentos de función/valores de retorno	No
x12–17	a2–7	Argumentos de función	No
x18–27	s2–11	Saved registers	Sí
x28–31	t3–6	Temporales	No
f0–7	ft0–7	Temporales, FP	No
f8–9	fs0–1	Saved registers, FP	Sí
f10–11	fa0–1	Argumentos/valores de retorno, FP	No
f12–17	fa2–7	Argumentos, FP	No
f18–27	fs2–11	Saved registers, FP	Sí
f28–31	ft8–11	Temporales, FP	No

Figura 3.2: Mnemónicos de ensamblador para registros enteros y de punto flotante en RISC-V. RISC-V tiene suficientes registros que el ABI puede reservar registros para ser usados por funciones hoja sin necesidad de guardarlos y restaurarlos. Los registros preservados a través de llamadas a funciones también son llamados *caller saved* versus *callee saved*, los cuales no se conservan. El Capítulo 5 explica los registros de punto flotante `f` (La Tabla 20.1 de [Waterman and Asanović 2017] es la base para esta figura).

Si hay demasiados argumentos y variables en la función para caber en los registros, el prólogo reserva espacio en el stack para la función, a esto se le llama *frame*. Luego que la función ha concluido, el *epílogo* restaura el stack frame y regresa al punto de origen:

```
# restaurar registros del stack de ser necesario
lw   ra,framesize-4(sp)  # Restaurar el registro con la dirección de retorno
addi sp,sp, framesize    # Liberar el espacio del stack frame
ret                      # Retornar a donde se invocó la función
```

Pronto veremos un ejemplo basado en este ABI, pero primero debemos explicar las demás tareas del ensamblador más allá de la conversión de nombres a números de registro.

Pseudoinstrucción	Instrucción/Instrucciones Base	Significado
`nop`	`addi x0, x0, 0`	No operation
`neg rd, rs`	`sub rd, x0, rs`	Complemento a 2
`negw rd, rs`	`subw rd, x0, rs`	Complemento a 2 (word)
`snez rd, rs`	`sltu rd, x0, rs`	Poner en 1 si $\neq$ cero
`sltz rd, rs`	`slt rd, rs, x0`	Poner en 1 si $<$ cero
`sgtz rd, rs`	`slt rd, x0, rs`	Poner en 1 si $>$ cero
`beqz rs, offset`	`beq rs, x0, offset`	Branch si $=$ cero
`bnez rs, offset`	`bne rs, x0, offset`	Branch si $\neq$ cero
`blez rs, offset`	`bge x0, rs, offset`	Branch si $\leq$ cero
`bgez rs, offset`	`bge rs, x0, offset`	Branch si $\geq$ cero
`bltz rs, offset`	`blt rs, x0, offset`	Branch si $<$ cero
`bgtz rs, offset`	`blt x0, rs, offset`	Branch si $>$ cero
`j offset`	`jal x0, offset`	Jump
`jr rs`	`jalr x0, rs, 0`	Jump a registro
`ret`	`jalr x0, x1, 0`	Retornar de subrutina
`tail offset`	`auipc x6, offset[31:12]` `jalr x0, x6, offset[11:0]`	*Tail call* subrutina lejana
`rdinstret[h] rd`	`csrrs rd, instret[h], x0`	Leer el contador de instrucciones retiradas
`rdcycle[h] rd`	`csrrs rd, cycle[h], x0`	Leer el contador de ciclos
`rdtime[h] rd`	`csrrs rd, time[h], x0`	Leer *real-time clock*
`csrr rd, csr`	`csrrs rd, csr, x0`	Leer CSR
`csrw csr, rs`	`csrrw x0, csr, rs`	Escribir CSR
`csrs csr, rs`	`csrrs x0, csr, rs`	Poner bits en 1 en CSR
`csrc csr, rs`	`csrrc x0, csr, rs`	Poner bits en 0 en CSR
`csrwi csr, imm`	`csrrwi x0, csr, imm`	Escribir CSR, inmediato
`csrsi csr, imm`	`csrrsi x0, csr, imm`	Poner bits en 1 en CSR, inmediato
`csrci csr, imm`	`csrrci x0, csr, imm`	Poner bits en 0 en CSR, inmediato
`frcsr rd`	`csrrs rd, fcsr, x0`	Leer *FP control/status register*
`fscsr rs`	`csrrw x0, fcsr, rs`	Escribir *FP control/status register*
`frrm rd`	`csrrs rd, frm, x0`	Leer *FP rounding mode*
`fsrm rs`	`csrrw x0, frm, rs`	Escribir *FP rounding mode*
`frflags rd`	`csrrs rd, fflags, x0`	Leer *FP exception flags*
`fsflags rs`	`csrrw x0, fflags, rs`	Escribir *FP exception flags*

Figura 3.3: 32 pseudo-instrucciones de RISC-V que dependen de x0, el registro cero. El Apéndice A incluye tanto las instrucciones reales como pseudoinstrucciones de RISC-V. Las que leen los contadores de 64 bits pueden leer los 32 bits de la parte alta utilizando la versión "h" de la instrucción y la versión normal para leer la parte baja (Las Tablas 20.2 y 20.3 de [Waterman and Asanović 2017] son la base de esta figura).

Pseudoinstrucción	Instrucción/Instrucciones Base	Significado
`lla rd, symbol`	`auipc rd, symbol[31:12]` `addi rd, rd, symbol[11:0]`	Load de dirección local
`la rd, symbol`	*PIC*: `auipc rd, GOT[symbol][31:12]` `l{w|d} rd, rd, GOT[symbol][11:0]` *No-PIC*: Igual que `lla rd, symbol`	Load de dirección
`l{b|h|w|d} rd, symbol`	`auipc rd, symbol[31:12]` `l{b|h|w|d} rd, symbol[11:0](rd)`	Load global
`s{b|h|w|d} rd, symbol, rt`	`auipc rt, symbol[31:12]` `s{b|h|w|d} rd, symbol[11:0](rt)`	Store global
`fl{w|d} rd, symbol, rt`	`auipc rt, symbol[31:12]` `fl{w|d} rd, symbol[11:0](rt)`	Load global de punto flotante
`fs{w|d} rd, symbol, rt`	`auipc rt, symbol[31:12]` `fs{w|d} rd, symbol[11:0](rt)`	Store global de punto flotante
`li rd, immediate`	*Muchas secuencias*	Load immediate
`mv rd, rs`	`addi rd, rs, 0`	Copiar registro
`not rd, rs`	`xori rd, rs, -1`	Complemento a uno
`sext.w rd, rs`	`addiw rd, rs, 0`	Sign extend word
`seqz rd, rs`	`sltiu rd, rs, 1`	Poner en 1 si = cero
`fmv.s rd, rs`	`fsgnj.s rd, rs, rs`	Copiar registro de precisión simple
`fabs.s rd, rs`	`fsgnjx.s rd, rs, rs`	Valor absoluto de precisión simple
`fneg.s rd, rs`	`fsgnjn.s rd, rs, rs`	Negación de precisión simple
`fmv.d rd, rs`	`fsgnj.d rd, rs, rs`	Copiar registro de precisión doble
`fabs.d rd, rs`	`fsgnjx.d rd, rs, rs`	Valor absoluto de precisión doble
`fneg.d rd, rs`	`fsgnjn.d rd, rs, rs`	Negación de precisión doble
`bgt rs, rt, offset`	`blt rt, rs, offset`	Branch si $>$
`ble rs, rt, offset`	`bge rt, rs, offset`	Branch si $\leq$
`bgtu rs, rt, offset`	`bltu rt, rs, offset`	Branch si $>$, unsigned
`bleu rs, rt, offset`	`bgeu rt, rs, offset`	Branch si $\leq$, unsigned
`jal offset`	`jal x1, offset`	Jump and link
`jalr rs`	`jalr x1, rs, 0`	Jump and link a registro
`call offset`	`auipc x1, offset[31:12]` `jalr x1, x1, offset[11:0]`	Llamar subrutina lejana
`fence`	`fence iorw, iorw`	*Fence* en toda la memoria e I/O
`fscsr rd, rs`	`csrrw rd, fcsr, rs`	Swap con *FP control/status register*
`fsrm rd, rs`	`csrrw rd, frm, rs`	Swap con *FP rounding mode*
`fsflags rd, rs`	`csrrw rd, fflags, rs`	Swap con *FP exception flags*

Figura 3.4: 28 pseudoinstrucciones de RISC-V que son independientes de x0, el registro cero. Para `la`, GOT significa Global Offset Table, y mantiene las direcciones de los símbolos en las librerías *linkeadas* dinámicamente. El Apéndice A incluye tanto las instrucciones reales como pseudoinstrucciones de RISC-V (Las Tablas 20.2 y 20.3 de [Waterman and Asanović 2017] son la base para esta figura).