RISC-V Reference Card

	Instruction	Name	Description	Type	Opcode	Funct3	Funct7
Arithmetic	**add rd rs1 rs2**	ADD	**rd = rs1 + rs2**	R	011 0011	000	000 0000
	sub rd rs1 rs2	SUBtract	**rd = rs1 - rs2**	R	011 0011	000	010 0000
	and rd rs1 rs2	bitwise AND	**rd = rs1 & rs2**	R	011 0011	111	000 0000
	or rd rs1 rs2	bitwise OR	**rd = rs1 | rs2**	R	011 0011	110	000 0000
	xor rd rs1 rs2	bitwise XOR	**rd = rs1 ^ rs2**	R	011 0011	100	000 0000
	sll rd rs1 rs2	Shift Left Logical	**rd = rs1 << rs2**	R	011 0011	001	000 0000
	srl rd rs1 rs2	Shift Right Logical	**rd = rs1 >> rs2** (Zero-extend)	R	011 0011	101	000 0000
	sra rd rs1 rs2	Shift Right Arithmetic	**rd = rs1 >> rs2** (Sign-extend)	R	011 0011	101	010 0000
	slt rd rs1 rs2	Set Less Than (signed)	**rd = (rs1 < rs2) ? 1 : 0**	R	011 0011	010	000 0000
	sltu rd rs1 rs2	Set Less Than (Unsigned)		R	011 0011	011	000 0000
	addi rd rs1 imm	ADD Immediate	**rd = rs1 + imm**	I	001 0011	000	
	andi rd rs1 imm	bitwise AND Immediate	**rd = rs1 & imm**	I	001 0011	111	
	ori rd rs1 imm	bitwise OR Immediate	**rd = rs1 | imm**	I	001 0011	110	
	xori rd rs1 imm	bitwise XOR Immediate	**rd = rs1 ^ imm**	I	001 0011	100	
	slli rd rs1 imm	Shift Left Logical Immediate	**rd = rs1 << imm**	I*	001 0011	001	000 0000
	srli rd rs1 imm	Shift Right Logical Immediate	**rd = rs1 >> imm** (Zero-extend)	I*	001 0011	101	000 0000
	srai rd rs1 imm	Shift Right Arithmetic Immediate	**rd = rs1 >> imm** (Sign-extend)	I*	001 0011	101	010 0000
	slti rd rs1 imm	Set Less Than Immediate (signed)	**rd = (rs1 < imm) ? 1 : 0**	I	001 0011	010	
	sltiu rd rs1 imm	Set Less Than Immediate (Unsigned)		I	001 0011	011	
Memory	**lb rd imm(rs1)**	Load Byte	**rd =** 1 byte of memory at address **rs1 + imm**, sign-extended	I	000 0011	000	
	lbu rd imm(rs1)	Load Byte (Unsigned)	**rd =** 1 byte of memory at address **rs1 + imm**, zero-extended	I	000 0011	100	
	lh rd imm(rs1)	Load Half-word	**rd =** 2 bytes of memory starting at address **rs1 + imm**, sign-extended	I	000 0011	001	
	lhu rd imm(rs1)	Load Half-word (Unsigned)	**rd =** 2 bytes of memory starting at address **rs1 + imm**, zero-extended	I	000 0011	101	
	lw rd imm(rs1)	Load Word	**rd =** 4 bytes of memory starting at address **rs1 + imm**	I	000 0011	010	
	sb rs2 imm(rs1)	Store Byte	Stores least-significant byte of **rs2** at the address **rs1 + imm** in memory	S	010 0011	000	
	sh rs2 imm(rs1)	Store Half-word	Stores the 2 least-significant bytes of **rs2** starting at the address **rs1 + imm** in memory	S	010 0011	001	
	sw rs2 imm(rs1)	Store Word	Stores **rs2** starting at the address **rs1 + imm** in memory	S	010 0011	010	

	Instruction	Name	Description	Type	Opcode	Funct3
Control	`beq rs1 rs2 label`	Branch if EQual	`if (rs1 == rs2)` `PC = PC + offset`	B	`110 0011`	`000`
	`bge rs1 rs2 label`	Branch if Greater or Equal (signed)	`if (rs1 >= rs2)` `PC = PC + offset`	B	`110 0011`	`101`
	`bgeu rs1 rs2 label`	Branch if Greater or Equal (Unsigned)		B	`110 0011`	`111`
	`blt rs1 rs2 label`	Branch if Less Than (signed)	`if (rs1 < rs2)` `PC = PC + offset`	B	`110 0011`	`100`
	`bltu rs1 rs2 label`	Branch if Less Than (Unsigned)		B	`110 0011`	`110`
	`bne rs1 rs2 label`	Branch if Not Equal	`if (rs1 != rs2)` `PC = PC + offset`	B	`110 0011`	`001`
	`jal rd label`	Jump And Link	`rd = PC + 4` `PC = PC + offset`	J	`110 1111`	
	`jalr rd rs1 imm`	Jump And Link Register	`rd = PC + 4` `PC = rs1 + imm`	I	`110 0111`	`000`
Other	`auipc rd imm`	Add Upper Immediate to PC	`rd = PC + (imm << 12)`	U	`001 0111`	
	`lui rd imm`	Load Upper Immediate	`rd = imm << 12`	U	`011 0111`	
	`ebreak`	Environment BREAK	Asks the debugger to do something (`imm = 0`)	I	`111 0011`	`000`
	`ecall`	Environment CALL	Asks the OS to do something (`imm = 1`)	I	`111 0011`	`000`
Ext	`mul rd rs1 rs2`	MULtiply (part of mul ISA extension)	`rd = rs1 * rs2`	(omitted)		

#	Name	Description	#	Name	Desc
`x0`	`zero`	Constant 0	`x16`	`a6`	*Args*
`x1`	`ra`	*Return Address*	`x17`	`a7`	
`x2`	`sp`	Stack Pointer	`x18`	`s2`	*Saved Registers*
`x3`	`gp`	Global Pointer	`x19`	`s3`	
`x4`	`tp`	Thread Pointer	`x20`	`s4`	
`x5`	`t0`	*Temporary Registers*	`x21`	`s5`	
`x6`	`t1`		`x22`	`s6`	
`x7`	`t2`		`x23`	`s7`	
`x8`	`s0`	Saved Registers	`x24`	`s8`	
`x9`	`s1`		`x25`	`s9`	
`x10`	`a0`	*Function Arguments or Return Values*	`x26`	`s10`	
`x11`	`a1`		`x27`	`s11`	
`x12`	`a2`	*Function Arguments*	`x28`	`t3`	*Temporaries*
`x13`	`a3`		`x29`	`t4`	
`x14`	`a4`		`x30`	`t5`	
`x15`	`a5`		`x31`	`t6`	

Caller saved registers

Callee saved registers (except `x0`, `gp`, `tp`)

Pseudoinstruction	Name	Description	Translation
`beqz rs1 label`	Branch if EQuals Zero	`if (rs1 == 0)` `PC = PC + offset`	`beq rs1 x0 label`
`bnez rs1 label`	Branch if Not Equals Zero	`if (rs1 != 0)` `PC = PC + offset`	`bne rs1 x0 label`
`j label`	Jump	`PC = PC + offset`	`jal x0 label`
`jr rs1`	Jump Register	`PC = rs1`	`jalr x0 rs1 0`
`la rd label`	Load absolute Address	`rd = &label`	`auipc`, `addi`
`li rd imm`	Load Immediate	`rd = imm`	`lui` (if needed), `addi`
`mv rd rs1`	MoVe	`rd = rs1`	`addi rd rs1 0`
`neg rd rs1`	NEGate	`rd = -rs1`	`sub rd x0 rs1`
`nop`	No OPeration	do nothing	`addi x0 x0 0`
`not rd rs1`	bitwise NOT	`rd = ~rs1`	`xori rd rs1 -1`
`ret`	RETurn	`PC = ra`	`jalr x0 x1 0`

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

Immediates are sign-extended to 32 bits, except in I* type instructions and `sltiu`.

Selected ASCII values

HEX	DEC	CHAR	HEX	DEC	CHAR	HEX	DEC	CHAR	HEX	DEC	CHAR	HEX	DEC	CHAR	HEX	DEC	CHAR
0x20	32	SPACE	0x30	48	0	0x40	64	@	0x50	80	P	0x60	96	`	0x70	112	p
0x21	33	!	0x31	49	1	0x41	65	A	0x51	81	Q	0x61	97	a	0x71	113	q
0x22	34	"	0x32	50	2	0x42	66	B	0x52	82	R	0x62	98	b	0x72	114	r
0x23	35	#	0x33	51	3	0x43	67	C	0x53	83	S	0x63	99	c	0x73	115	s
0x24	36	$	0x34	52	4	0x44	68	D	0x54	84	T	0x64	100	d	0x74	116	t
0x25	37	%	0x35	53	5	0x45	69	E	0x55	85	U	0x65	101	e	0x75	117	u
0x26	38	&	0x36	54	6	0x46	70	F	0x56	86	V	0x66	102	f	0x76	118	v
0x27	39	'	0x37	55	7	0x47	71	G	0x57	87	W	0x67	103	g	0x77	119	w
0x28	40	(	0x38	56	8	0x48	72	H	0x58	88	X	0x68	104	h	0x78	120	x
0x29	41	)	0x39	57	9	0x49	73	I	0x59	89	Y	0x69	105	i	0x79	121	y
0x2A	42	*	0x3A	58	:	0x4A	74	J	0x5A	90	Z	0x6A	106	j	0x7A	122	z
0x2B	43	+	0x3B	59	;	0x4B	75	K	0x5B	91	[	0x6B	107	k	0x7B	123	{
0x2C	44	,	0x3C	60	<	0x4C	76	L	0x5C	92	\	0x6C	108	l	0x7C	124	|
0x2D	45	-	0x3D	61	=	0x4D	77	M	0x5D	93	]	0x6D	109	m	0x7D	125	}
0x2E	46	.	0x3E	62	>	0x4E	78	N	0x5E	94	^	0x6E	110	n	0x7E	126	~
0x2F	47	/	0x3F	63	?	0x4F	79	O	0x5F	95	_	0x6F	111	o	0x00	0	NULL

C Format String Specifiers

Specifier	Output
`d` or `i`	Signed decimal integer
`u`	Unsigned decimal integer
`o`	Unsigned octal
`x`	Unsigned hexadecimal integer, lowercase
`X`	Unsigned hexadecimal integer, uppercase
`f`	Decimal floating point, lowercase
`F`	Decimal floating point, uppercase
`e`	Scientific notation (significand/exponent), lowercase
`E`	Scientific notation (significand/exponent), uppercase
`g`	Use the shortest representation: `%e` or `%f`
`G`	Use the shortest representation: `%E` or `%F`
`a`	Hexadecimal floating point, lowercase
`A`	Hexadecimal floating point, uppercase
`c`	Character
`s`	String of characters
`p`	Pointer address

IEEE 754 Floating Point Standard

	Sign	Exponent	Significand
Single Precision	1 bit	8 bits (bias = -127)	23 bits
Double Precision	1 bit	11 bits (bias = -1023)	52 bits
Quad Precision	1 bit	15 bits (bias = -16383)	112 bits

Standard exponent bias: - (2^{E-1}-1) where E is the number of exponent bits

SI Prefixes

Size	Prefix	Symbol	Size	Prefix	Symbol	Size	Prefix	Symbol
10^{-3}	milli-	m	10^{3}	kilo-	k	2^{10}	kibi-	Ki
10^{-6}	micro-	μ	10^{6}	mega-	M	2^{20}	mebi-	Mi
10^{-9}	nano-	n	10^{9}	giga-	G	2^{30}	gibi-	Gi
10^{-12}	pico-	p	10^{12}	tera-	T	2^{40}	tebi-	Ti
10^{-15}	femto-	f	10^{15}	peta-	P	2^{50}	pebi-	Pi
10^{-18}	atto-	a	10^{18}	exa-	E	2^{60}	exbi-	Ei
10^{-21}	zepto-	z	10^{21}	zetta-	Z	2^{70}	zebi-	Zi
10^{-24}	yocto-	y	10^{24}	yotta-	Y	2^{80}	yobi-	Yi

Laws of Boolean Algebra

$$x \cdot \overline{x} = 0 \qquad x + \overline{x} = 1 \qquad (xy)z = x\,(yz)$$

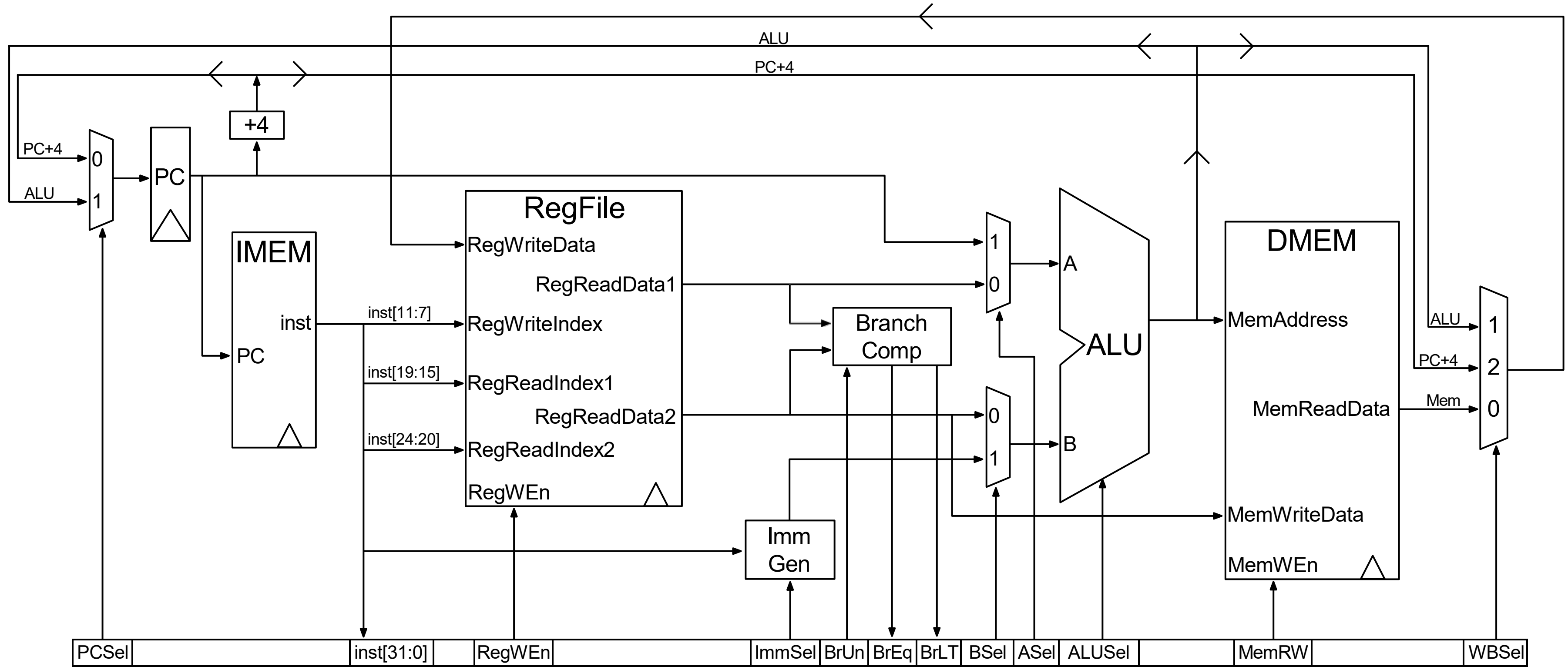
$$x \cdot 0 = 0 \qquad x + 1 = 1 \qquad (x+y)+z = x+(y+z)$$

$$x \cdot 1 = x \qquad x + 0 = x \qquad x\,(y+z) = xy + xz$$

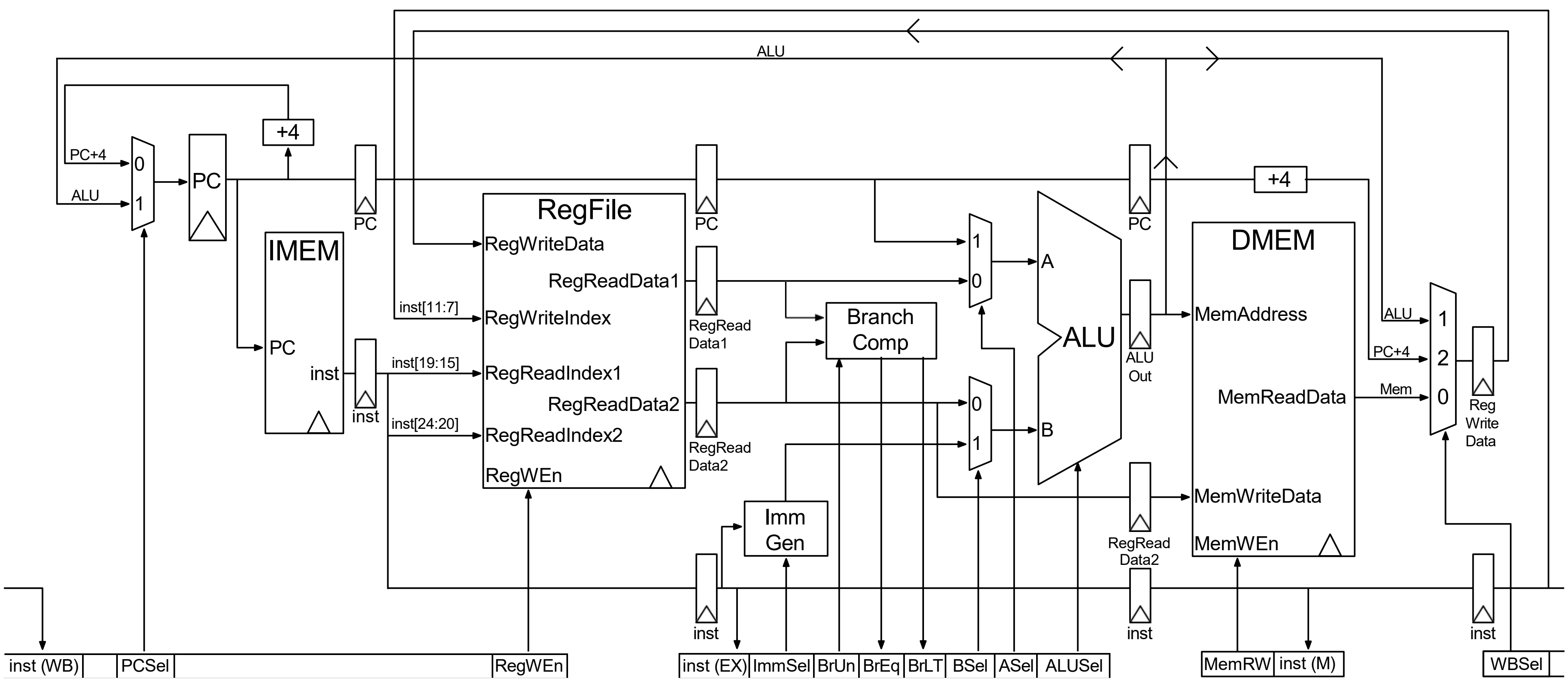
$$x \cdot x = x \qquad x + x = x \qquad x + yz = (x+y)\,(x+z)$$

$$x \cdot y = y \cdot x \qquad x + y = y + x \qquad \overline{x \cdot y} = \overline{x} + \overline{y}$$

$$xy + x = x \qquad (x+y)x = x \qquad \overline{(x+y)} = \overline{x} \cdot \overline{y}$$



ALU
PC+4
PC+4
ALU
0
1
PC
+4
IMEM
inst
PC
RegFile
RegWriteData
RegReadData1
RegWriteIndex
RegReadIndex1
RegReadData2
RegReadIndex2
RegWEn
inst[11:7]
inst[19:15]
inst[24:20]
Branch
Comp
Imm
Gen
1
0
0
1
A
B
ALU
DMEM
MemAddress
MemReadData
MemWriteData
MemWEn
ALU
PC+4
Mem
1
2
0
PCSel
inst[31:0]
RegWEn
ImmSel
BrUn
BrEq
BrLT
BSel
ASel
ALUSel
MemRW
WBSel



ALU
PC+4
ALU
0
1
PC
+4
PC
IMEM
PC
inst
inst
RegFile
RegWriteData
RegReadData1
inst[11:7]
RegWriteIndex
inst[19:15]
RegReadIndex1
RegReadData2
inst[24:20]
RegReadIndex2
RegWEn
PC
RegRead Data1
RegRead Data2
Branch Comp
Imm Gen
inst
1
0
0
1
A
B
ALU
PC
ALU Out
RegRead Data2
inst
+4
DMEM
MemAddress
MemReadData
MemWriteData
MemWEn
ALU
PC+4
Mem
1
2
0
Reg Write Data
inst
inst (WB)
PCSel
RegWEn
inst (EX)
ImmSel
BrUn
BrEq
BrLT
BSel
ASel
ALUSel
MemRW
inst (M)
WBSel