Opcode	Mnemonic	Operands Instr	Operand Stack before	Operand Stack after	Description
0x32	aaload		…, arrayref, index	…, value	load a reference stored in an array and put it on the stack
0x53	aastore		…, arrayref, index, value	…	store a reference in an array
0x01	aconst_null		…	…, null	push a null-reference on the stack
0x19	aload	index	…	…, objectref	load a reference from local variable
0x2a	aload_0		…	…, objectref	load a reference from local variable with index 0
0x2b	aload_1		…	…, objectref	index 1
0x2c	aload_2		…	…, objectref	index 2
0x2d	aload_3		…	…, objectref	index 3
0xbd	anewarray	indexbyte1, indexbyte2	…, count	…, arrayref	create a new array of type cp[index] with count items
0xb0	areturn		…, objectref	[empty]	return a reference to the calling method
0xbe	arraylength		…, objectref	…, length	return the length of an array
0x3a	astore	index	…, objectref	…	store a reference in a local variable
0x4b	astore_0		…, objectref	…	store a reference in a local variable with index 0
0x4c	astore_1		…, objectref	…	index 1
0x4d	astore_2		…, objectref	…	index 2
0x4e	astore_3		…, objectref	…	index 3
0xbf	athrow		…, objectref	objectref	throw the exception of objectref from stack
0x33	baload		…, arrayref, index	…, value	load a byte or boolean from array
0x54	bastore		…, arrayref, index, value	…	store a byte or boolean in an array
0x10	bipush	byte	…	…, value	store the byte with sign extended as int to the stack
0x34	caload		…, arrayref, index	…, value	load a char from array
0x55	castore		…, arrayref, index, value	…	store a char to an array
0xc0	checkcast	indexbyte1, indexbyte2	…, objectref	…, objectref	check if reference is of type cp[index], throw ClassCastException
0x90	d2f		…, value	…, result	double to float
0x8e	d2i		…, value	…, result	double to int (NaN = 0)
0x8f	d2l		…, value	…, result	double to long
0x63	dadd		…, value1, value2	…, result	double add
0x31	daload		…, arrayref, index	…, value	load double from array
0x52	dastore		…, arrayref, index, value	…	store double in array
0x98	dcmpg		…, value1, value2	…, result	(int) 1 <= v1>v2; 0 <= v1=v2; -1 v1<v2; NaN = 1
0x97	dcmpl		…, value1, value2	…, result	(int) Nan = -1
0x0e	dconst_0		…	…, 0.0	push double 0.0 to stack
0x0f	dconst_1		…	…, 1.0	push double 1.0 to stack
0x6f	ddiv		…, value1, value2	…, result	double division, result is double
0x18	dload	index	…	…, value	load double from local variable
0x26	dload_0		…	…, value	load doube from local variable with index 0
0x27	dload_1		…	…, value	index 1
0x28	dload_2		…	…, value	index 2
0x29	dload_3		…	…, value	index 3
0x6b	dmul		…, value1, value2	…, result	doulbe multiplication, result is double
0x77	dneg		…, value	…, result	result <= - value
0x73	drem		…, value1, value2	…, result	calc the reminder of the division as double
0xaf	dreturn		…, value	[empty]	return a double to the calling method
0x39	dstore	index	…, value	…	store a double form the stack to an local variable
0x47	dstore_0		…, value	…	store a double form the stack to an local variable with index 0
0x48	dstore_1		…, value	…	index 1
0x49	dstore_2		…, value	…	index 2
0x4a	dstore_3		…, value	…	index 3
0x67	dsub		…, value1, value2	…, result	subtract doubles
0x59	dup		…, value	…, value, value	duplicate top of stack (TOS)
0x5a	dup_x1		…, value2, value1	…, value1, value2, value1	dup ToS two values down
0x5b	dup_x2		…, value3, value2, value1	…, value1, value3, value2, value1	dup Tos three values down, v1..v3 are category 1
0x5b			…, value2, value1	…, value1, value2, value1	dup ToS two values down, v1 is category 1, v2 is category 2 (long, double)
0x5c	dup2		…, value2, value1	…, value2, value1, value2, value1	dup two Tos values (category 1)
0x5c			…, value	…, value, value	dup one Tos value (category 2)
0x5d	dup2_x1		..., value3, value2, value1	…, value2, value1, value3, value2, value1	v1..v3 are of category 1
0x5d			…, value2, value1	…, value1, value2, value1	v1 category 2 and v2 is category 1
0x5e	dup2_x2		…, v4, v3, v2, v1	…, v2, v1, v4, v3, v2, v1	v1..v4 are of category 1
0x5e			…, v3, v2, v1	…, v1, v3, v2, v1	v1 category 2 and v2, v3 is category 1
0x5e			…, v2, v1	…, v1, v2, v1	v1 and v2 are category 2
0x8d	f2d		…, value	…, result	float to double
0x8b	f2i		…, value	…, result	float to int
0x8c	f2l		…, value	…, result	float to long
0x62	fadd		…, value1, value2	…, result	float add
0x30	faload		…, arrayref, index	…, value	load float from array
0x51	fastore		…, arrayref, index, value	…	store float to array
0x96	fcmpg		…, value1, value2	…, result	(int) 1 <= v1>v2; 0 <= v1=v2; -1 v1<v2; NaN = 1
0x95	fcmpl		…, value1, value2	…, result	(int) Nan = -1
0x0b	fconst_0		…	…, 0.0	push float 0.0 to stack
0x0d	fconst_1		…	…, 1.0	push float 1.0 to stack
0x6e	fdiv		…, value1, value2	…, result	float division, result is float
0x17	fload	index	…	…, value	load float from local variable

Opcode	Mnemonic	Operands Instr	Operand Stack before	Operand Stack after	Description
0x22	fload_0		…	…, value	load float from local variable with index 0
0x23	fload_1		…	…, value	index 1
0x24	fload_2		…	…, value	index 2
0x25	fload_3		…	…, value	index 3
0x6a	fmul		…, value1, value2	…, result	float multiplication, result is float
0x76	fneg		…, value	…, result	result <= - value
0x72	frem		…, value1, value2	…, result	calc the reminder of the division as float
0xae	freturn		…, value	[empty]	return a float to the calling method
0x38	fstore	index	…, value	…	store a float form the stack to an local variable
0x43	fstore_0		…, value	…	store a float form the stack to an local variable with index 0
0x44	fstore_1		…, value	…	index 1
0x45	fstore_2		…, value	…	index 2
0x46	fstore_3		…, value	…	index 3
0x66	fsub		…, value1, value2	…, result	subtract floats
0xb4	getfield	indexbyte1, indexbyte2	…, objectref	…, value	get the value of a field
0xb2	getstatic	indexbyte1, indexbyte2	…	…, value	get the value of a static field
0xa7	goto	branchbyte1, branchbyte2			jump to the method instruction at position branch
0xc8	goto_w	branchbyte1..branchbyte4			jump to the method instruction at position branch, for future use?
0x91	i2b		…, value	…, result	integer to byte; result <= signextend(value & 0xff)
0x92	i2c		…, value	…, result	integer to char; result <= zeroextend(value & 0xffff)
0x87	i2d		…, value	…, result	integer to double
0x86	i2f		…, value	…, result	integer to float
0x85	i2l		…, value	…, result	integer to long, result <= signextend_long(value)
0x93	i2s		…, value	…, result	integer to short; result <= singextend(value & 0xffff)
0x60	iadd		…, value1, value2	…, result	add int
0x2e	iaload		…, arrayref, index	…, value	load int from array
0x7e	iand		…, value1, value2	…, result	bitwise AND int
0x4f	iastore		…, arrayref, index, value	…	store int to array
0x02	iconst_m1		…	…, -1	push -1 to stack
0x03	iconst_0		…	…, 0	push 0 to stack
0x04	iconst_1			…, 1	push 1 to stack
0x05	iconst_2			…, 2	push 2 to stack
0x06	iconst_3			…, 3	push 3 to stack
0x07	iocnst_4			…, 4	push 4 to stack
0x08	iconst_5			…, 5	push 5 to stack
0x6c	idiv		…, value1, value2	…, result	int division, result is int, round to 0, truncate
0xa5	if_acmpeq	branchbyte1, branchbyte2	…, value1, value2	…	branch if reference is equal. "branch" is a 16-bit offset
0xa6	if_acmpne	branchbyte1, branchbyte2	…, value1, value2	…	branch if reference is not equal. "branch" is a 16-bit offset
0x9f	if_icmp_eq	branchbyte1, branchbyte2	…, value1, value2	…	branch if value1=value2, "branch" is a 16-bit offset
0xa0	if_icmp_ne	branchbyte1, branchbyte2	…, value1, value2	…	branch if value1!=value2, "branch" is a 16-bit offset
0xa1	if_icmp_lt	branchbyte1, branchbyte2	…, value1, value2	…	branch if value1<value2, "branch" is a 16-bit offset
0xa2	if_icmp_ge	branchbyte1, branchbyte2	…, value1, value2	…	branch if value1>=value2, "branch" is a 16-bit offset
0xa3	if_icmp_gt	branchbyte1, branchbyte2	…, value1, value2	…	branch if value1>value2, "branch" is a 16-bit offset
0xa4	if_icmp_le	branchbyte1, branchbyte2	…, value1, value2	…	branch if value1<=value2, "branch" is a 16-bit offset
0x99	ifeq	branchbyte1, branchbyte2	…, value	…	branch if value = 0, "branch" is a 16-bit offset
0x9a	ifne	branchbyte1, branchbyte2	…, value	…	branch if value != 0, "branch" is a 16-bit offset
0x9b	iflt	branchbyte1, branchbyte2	…, value	…	branch if value < 0, "branch" is a 16-bit offset
0x9c	ifge	branchbyte1, branchbyte2	…, value	…	branch if value >= 0, "branch" is a 16-bit offset
0x9d	ifgt	branchbyte1, branchbyte2	…, value	…	branch if value > 0, "branch" is a 16-bit offset
0x9e	ifle	branchbyte1, branchbyte2	…, value	…	branch if value <= 0, "branch" is a 16-bit offset
0xc7	ifnonnull	branchbyte1, branchbyte2	…, value	…	branch if reference != null
0xc6	ifnull	branchbyte1, branchbyte2	…, value	…	branch if reference = null
0x84	iinc	index, const			inc local variable by constant
0x15	iload	index	…	…, value	load int from local variable
0x1a	iload_0		…	…, value	load int from local variable with index 0
0x1b	iload_1		…	…, value	index 1
0x1c	iload_2		…	…, value	index 2
0x1d	iload_3		…	…, value	index 3
0x68	imul		…, value1, value2	…, result	int multiplication, result is int
0x74	ineg		…, value	…, result	result <= - value
0xc1	instanceof	indexbyte1, indexbyte2	…, objectref	…, result	check if objectref is an instance of the type in the cp[index] (result =1,0)
0xb9	invokeinterface	indexbyte1, indexbyte2, count, 0	…, objectref, [arg1, …]	…	invoke interface method
0xb7	invokespecial	indexbyte1, indexbyte2	…, objectref, [arg1, …]	…	invoke instance method, handling superclass, private, instance init
0xb8	invokestatic	indexbyte1, indexbyte2	…, [arg1, …]	…	invoke a class static method
0xb6	invokevirtual	indexbyte1, indexbyte2	…, objectref, [arg1, …]		invoke a instance method, dispatch based on class
0x80	ior		…, value1, value2	…, result	bitwise inclusive or
0x70	irem		…, value1, value2	…, result	calc the reminder of the division as int
0xac	ireturn		…, value	[empty]	return a int to the calling method
0x78	ishl		…, value1, value2	…, result	result <= value << (value2 && 0x1f)
0x7a	ishr		…, value1, value2	…, result	result <= value >> (value2 && 0x1f), with sign extension
0x36	istore	index	…, value	…	store a int form the stack to an local variable
0x3b	istore_0		…, value	…	store a int form the stack to an local variable with index 0

Opcode	Mnemonic	Operands Instr	Operand Stack before	Operand Stack after	Description
0x3c	istore_1		…, value	…	index 1
0x3d	istore_2		…, value	…	index 2
0x3e	istore_3		…, value	…	index 3
0x64	isub		…, value1, value2	…, result	subtract ints
0x7c	iushr		…, value1, value2	…, result	result <= value >> (value2 && 0x1f), with zero extension
0x82	ixor		…, value1, value2	…, result	bitwise exclusive or
0xa8	jsr	branchbyte1, branchbyte2	…	…, address	jump subroutine, address is instruction following the jsr as returnAddress
0xc9	jsr_w	branchbyte1..branchbyte4	…	…, address	jump subroutine, address is instruction following the jsr as returnAddress
0x8a	l2d		…, value	…, result	long to double
0x89	l2f		…, value	…, result	long to float
0x88	l2i		…, value	…, result	long to int
0x61	ladd		…, value1, value2	…, result	long add
0x2f	laload		…, arrayref, index	…, value	load long from array
0x7f	land		…, value1, value2	…, result	bitwise and long
0x50	lastore		…, arrayref, index, value	…	store long to array
0x94	lcmp		…, value1, value2	…, result	(int) 1 <= v1>v2; 0 <= v1=v2; -1 <= v1<v2
0x09	lconst_0		…	…, 0L	push long 0L to stack
0x0a	lconst_1		…	…, 1L	push long 1L to stack
0x12	ldc	index	…	…, value	load item from runtime constant pool and push it on the stack
0x13	ldc_w	indexbyte1, indexbyte2	…	…, value	load item from runtime constant pool and push it on the stack, if index>255
0x14	ldc2_w	indexbyte1, indexbyte2	…	…, value	load long/double item from runtime constant pool
0x6d	ldiv		…, value1, value2	…, result	long division, result is long
0x16	lload	index	…	…, value	load long from local variable
0x1e	lload_0		…	…, value	load long from local variable with index 0
0x1f	lload_1		…	…, value	index 1
0x20	lload_2		…	…, value	index 2
0x21	lload_3		…	…, value	index 3
0x69	lmul		…, value1, value2	…, result	long multiplication, result is long
0x75	lneg		…, value	…, result	result <= - value
0xab	lookupswitch	pad0..3, default1..4; npairs1..4; match-offset pairs…	…, key	….	Acess jump talbe by key match and jump
0x81	lor		…, value1, value2	…, result	bitwise inclusive or for longs
0x71	lrem		…, value1, value2	…, result	calc the reminder of the division as long
0xad	lreturn		…, value	[empty]	return a long to the calling method
0x79	lshl		…, value1, value2	…, result	long shift left; result = value1<<(value2 and 0x3f)
0x7b	lshr		…, value1, value2	…, result	arith shift left; result = value1>>(value2 and 0x3f); sign extension
0x37	lstore	index	…, value	…	store a long form the stack to an local variable
0x3f	lstore_0		…, value	…	store a long form the stack to an local variable with index 0,1
0x40	lstore_1		…, value	…	index 1,2
0x41	lstore_2		…, value	…	index 2,3
0x42	lstore_3		…, value	…	index 3,4
0x65	lsub		…, value1, value2	…, result	subtract longs
0x7d	lushr		…, value1, value2	…, result	result <= value >> (value2 && 0x3f), with zero extension
0x83	lxor		…, value1, value2	…, result	bitwise exclusive or
0xc2	monitorenter		…, objectref	…	enter monitor for object
0xc3	monitorexit		…, objectref	…	exit monitor for object
0xc5	multinewarray	indexbyte1, indexbyte2, dims	…, count1, [count2, …]	…, arrayref	create a multidimensional array with count_x elements in each dimension
0xbb	new	indexbyte1, indexbyte2	…	…, objectref	create a new object, defined in cp[index]
0xbc	newarray	atype	…, count	…, arrayref	create a new array, atype = {T_BOOLEAN=4; T_CHAR; T_FLOAT; T_DOUBLE; T_BYTE; T_SHORT; T_INT; T_LONG}
0x00	nop				nop
0x57	pop		…, value	…	pop ToS, value is category 1
0x58	pop2		…, value2, value1	…	pop two values of category 1 from ToS
0x58	pop2		…, value	…	pop one value of category 2 from ToS
0xb5	putfield	indexbyte1, indexbyte2	…, objectref, value	…	set the field in the objectref to value
0xb3	putstatic	indexbyte1, indexbyte2	…, value	…	set a static field in a class to value
0xa9	ret	index			return from subroutine, with jsr, jsr_w, for "finally"
0xb1	return		…	[empty]	return void from method
0x35	saload		…, arrayref, index	…, value	load a short from array
0x56	sastore		…, arrayref, index, value	…	store short in array
0x11	sipush	byte1, byte2	…	…, value	push sign extended byte1<<8 | byte2 as int to the stack
0x5f	swap		…, value2, value1	…, value1, value2	swap the top two operand on stack, value1 and value2 are of category 1
0xaa	tableswitch	pad0..3, default1..4; lowbyte1..4, highbyte1..4, jump_offset…	…, index	….	access jump table by index and jump
0xc4	wide	<opcode>, indexbyte1, indexbyte2	same as <opcode>	same as <opcode>	extend local variable index by additional bytes (local vars with index > 255) *load; *store; ret
0xc4	wide	iinc, indexbyte1, indexbyte2, constbyte1, constbyte2	same as <opcode>	same as <opcode>	inc local variable by constant >255
0xca	breakpoint				breakpoint for debugging
0xfe	impdep1				implementation defined (user opcode)
0xff	impdep2				implementation defined (user opcode)
0xba					unused