|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Mnemonic** | **Opcode *(in*** [***hex***](https://en.wikipedia.org/wiki/Hexadecimal)***)*** | **Opcode (in binary)** | **Other bytes [count]: [operand labels]** | **Stack [before]→[after]** | **Description** |
| aaload | 32 | 0011 0010 |  | arrayref, index → value | load onto the stack a reference from an array |
| aastore | 53 | 0101 0011 |  | arrayref, index, value → | store a reference in an array |
| aconst\_null | 01 | 0000 0001 |  | → null | push a *null* reference onto the stack |
| aload | 19 | 0001 1001 | 1: index | → objectref | load a reference onto the stack from a local variable *#index* |
| aload\_0 | 2a | 0010 1010 |  | → objectref | load a reference onto the stack from local variable 0 |
| aload\_1 | 2b | 0010 1011 |  | → objectref | load a reference onto the stack from local variable 1 |
| aload\_2 | 2c | 0010 1100 |  | → objectref | load a reference onto the stack from local variable 2 |
| aload\_3 | 2d | 0010 1101 |  | → objectref | load a reference onto the stack from local variable 3 |
| anewarray | bd | 1011 1101 | 2: indexbyte1, indexbyte2 | count → arrayref | create a new array of references of length *count* and component type identified by the class reference *index* (indexbyte1 << 8 + indexbyte2) in the constant pool |
| areturn | b0 | 1011 0000 |  | objectref → [empty] | return a reference from a method |
| arraylength | be | 1011 1110 |  | arrayref → length | get the length of an array |
| astore | 3a | 0011 1010 | 1: index | objectref → | store a reference into a local variable *#index* |
| astore\_0 | 4b | 0100 1011 |  | objectref → | store a reference into local variable 0 |
| astore\_1 | 4c | 0100 1100 |  | objectref → | store a reference into local variable 1 |
| astore\_2 | 4d | 0100 1101 |  | objectref → | store a reference into local variable 2 |
| astore\_3 | 4e | 0100 1110 |  | objectref → | store a reference into local variable 3 |
| athrow | bf | 1011 1111 |  | objectref → [empty], objectref | throws an error or exception (notice that the rest of the stack is cleared, leaving only a reference to the Throwable) |
| baload | 33 | 0011 0011 |  | arrayref, index → value | load a byte or Boolean value from an array |
| bastore | 54 | 0101 0100 |  | arrayref, index, value → | store a byte or Boolean value into an array |
| bipush | 10 | 0001 0000 | 1: byte | → value | push a *byte* onto the stack as an integer *value* |
| breakpoint | ca | 1100 1010 |  |  | reserved for breakpoints in Java debuggers; should not appear in any class file |
| caload | 34 | 0011 0100 |  | arrayref, index → value | load a char from an array |
| castore | 55 | 0101 0101 |  | arrayref, index, value → | store a char into an array |
| checkcast | c0 | 1100 0000 | 2: indexbyte1, indexbyte2 | objectref → objectref | checks whether an *objectref* is of a certain type, the class reference of which is in the constant pool at *index* (indexbyte1 << 8 + indexbyte2) |
| d2f | 90 | 1001 0000 |  | value → result | convert a double to a float |
| d2i | 8e | 1000 1110 |  | value → result | convert a double to an int |
| d2l | 8f | 1000 1111 |  | value → result | convert a double to a long |
| dadd | 63 | 0110 0011 |  | value1, value2 → result | add two doubles |
| daload | 31 | 0011 0001 |  | arrayref, index → value | load a double from an array |
| dastore | 52 | 0101 0010 |  | arrayref, index, value → | store a double into an array |
| dcmpg | 98 | 1001 1000 |  | value1, value2 → result | compare two doubles |
| dcmpl | 97 | 1001 0111 |  | value1, value2 → result | compare two doubles |
| dconst\_0 | 0e | 0000 1110 |  | → 0.0 | push the constant *0.0* (a *double*) onto the stack |
| dconst\_1 | 0f | 0000 1111 |  | → 1.0 | push the constant *1.0* (a *double*) onto the stack |
| ddiv | 6f | 0110 1111 |  | value1, value2 → result | divide two doubles |
| dload | 18 | 0001 1000 | 1: index | → value | load a double *value* from a local variable *#index* |
| dload\_0 | 26 | 0010 0110 |  | → value | load a double from local variable 0 |
| dload\_1 | 27 | 0010 0111 |  | → value | load a double from local variable 1 |
| dload\_2 | 28 | 0010 1000 |  | → value | load a double from local variable 2 |
| dload\_3 | 29 | 0010 1001 |  | → value | load a double from local variable 3 |
| dmul | 6b | 0110 1011 |  | value1, value2 → result | multiply two doubles |
| dneg | 77 | 0111 0111 |  | value → result | negate a double |
| drem | 73 | 0111 0011 |  | value1, value2 → result | get the remainder from a division between two doubles |
| dreturn | af | 1010 1111 |  | value → [empty] | return a double from a method |
| dstore | 39 | 0011 1001 | 1: index | value → | store a double *value* into a local variable *#index* |
| dstore\_0 | 47 | 0100 0111 |  | value → | store a double into local variable 0 |
| dstore\_1 | 48 | 0100 1000 |  | value → | store a double into local variable 1 |
| dstore\_2 | 49 | 0100 1001 |  | value → | store a double into local variable 2 |
| dstore\_3 | 4a | 0100 1010 |  | value → | store a double into local variable 3 |
| dsub | 67 | 0110 0111 |  | value1, value2 → result | subtract a double from another |
| dup | 59 | 0101 1001 |  | value → value, value | duplicate the value on top of the stack |
| dup\_x1 | 5a | 0101 1010 |  | value2, value1 → value1, value2, value1 | insert a copy of the top value into the stack two values from the top. value1 and value2 must not be of the type double or long. |
| dup\_x2 | 5b | 0101 1011 |  | value3, value2, value1 → value1, value3, value2, value1 | insert a copy of the top value into the stack two (if value2 is double or long it takes up the entry of value3, too) or three values (if value2 is neither double nor long) from the top |
| dup2 | 5c | 0101 1100 |  | {value2, value1} → {value2, value1}, {value2, value1} | duplicate top two stack words (two values, if value1 is not double nor long; a single value, if value1 is double or long) |
| dup2\_x1 | 5d | 0101 1101 |  | value3, {value2, value1} → {value2, value1}, value3, {value2, value1} | duplicate two words and insert beneath third word (see explanation above) |
| dup2\_x2 | 5e | 0101 1110 |  | {value4, value3}, {value2, value1} → {value2, value1}, {value4, value3}, {value2, value1} | duplicate two words and insert beneath fourth word |
| f2d | 8d | 1000 1101 |  | value → result | convert a float to a double |
| f2i | 8b | 1000 1011 |  | value → result | convert a float to an int |
| f2l | 8c | 1000 1100 |  | value → result | convert a float to a long |
| fadd | 62 | 0110 0010 |  | value1, value2 → result | add two floats |
| faload | 30 | 0011 0000 |  | arrayref, index → value | load a float from an array |
| fastore | 51 | 0101 0001 |  | arrayref, index, value → | store a float in an array |
| fcmpg | 96 | 1001 0110 |  | value1, value2 → result | compare two floats |
| fcmpl | 95 | 1001 0101 |  | value1, value2 → result | compare two floats |
| fconst\_0 | 0b | 0000 1011 |  | → 0.0f | push *0.0f* on the stack |
| fconst\_1 | 0c | 0000 1100 |  | → 1.0f | push *1.0f* on the stack |
| fconst\_2 | 0d | 0000 1101 |  | → 2.0f | push *2.0f* on the stack |
| fdiv | 6e | 0110 1110 |  | value1, value2 → result | divide two floats |
| fload | 17 | 0001 0111 | 1: index | → value | load a float *value* from a local variable *#index* |
| fload\_0 | 22 | 0010 0010 |  | → value | load a float *value* from local variable 0 |
| fload\_1 | 23 | 0010 0011 |  | → value | load a float *value* from local variable 1 |
| fload\_2 | 24 | 0010 0100 |  | → value | load a float *value* from local variable 2 |
| fload\_3 | 25 | 0010 0101 |  | → value | load a float *value* from local variable 3 |
| fmul | 6a | 0110 1010 |  | value1, value2 → result | multiply two floats |
| fneg | 76 | 0111 0110 |  | value → result | negate a float |
| frem | 72 | 0111 0010 |  | value1, value2 → result | get the remainder from a division between two floats |
| freturn | ae | 1010 1110 |  | value → [empty] | return a float |
| fstore | 38 | 0011 1000 | 1: index | value → | store a float *value* into a local variable *#index* |
| fstore\_0 | 43 | 0100 0011 |  | value → | store a float *value* into local variable 0 |
| fstore\_1 | 44 | 0100 0100 |  | value → | store a float *value* into local variable 1 |
| fstore\_2 | 45 | 0100 0101 |  | value → | store a float *value* into local variable 2 |
| fstore\_3 | 46 | 0100 0110 |  | value → | store a float *value* into local variable 3 |
| fsub | 66 | 0110 0110 |  | value1, value2 → result | subtract two floats |
| getfield | b4 | 1011 0100 | 2: indexbyte1, indexbyte2 | objectref → value | get a field *value* of an object *objectref*, where the field is identified by field reference in the constant pool *index* (indexbyte1 << 8 + indexbyte2) |
| getstatic | b2 | 1011 0010 | 2: indexbyte1, indexbyte2 | → value | get a static field *value* of a class, where the field is identified by field reference in the constant pool *index* (indexbyte1 << 8 + indexbyte2) |
| goto | a7 | 1010 0111 | 2: branchbyte1, branchbyte2 | [no change] | goes to another instruction at *branchoffset* (signed short constructed from unsigned bytes branchbyte1 << 8 + branchbyte2) |
| goto\_w | c8 | 1100 1000 | 4: branchbyte1, branchbyte2, branchbyte3, branchbyte4 | [no change] | goes to another instruction at *branchoffset* (signed int constructed from unsigned bytes branchbyte1 << 24 + branchbyte2 << 16 + branchbyte3 << 8 + branchbyte4) |
| i2b | 91 | 1001 0001 |  | value → result | convert an int into a byte |
| i2c | 92 | 1001 0010 |  | value → result | convert an int into a character |
| i2d | 87 | 1000 0111 |  | value → result | convert an int into a double |
| i2f | 86 | 1000 0110 |  | value → result | convert an int into a float |
| i2l | 85 | 1000 0101 |  | value → result | convert an int into a long |
| i2s | 93 | 1001 0011 |  | value → result | convert an int into a short |
| iadd | 60 | 0110 0000 |  | value1, value2 → result | add two ints |
| iaload | 2e | 0010 1110 |  | arrayref, index → value | load an int from an array |
| iand | 7e | 0111 1110 |  | value1, value2 → result | perform a bitwise AND on two integers |
| iastore | 4f | 0100 1111 |  | arrayref, index, value → | store an int into an array |
| iconst\_m1 | 02 | 0000 0010 |  | → -1 | load the int value −1 onto the stack |
| iconst\_0 | 03 | 0000 0011 |  | → 0 | load the int value 0 onto the stack |
| iconst\_1 | 04 | 0000 0100 |  | → 1 | load the int value 1 onto the stack |
| iconst\_2 | 05 | 0000 0101 |  | → 2 | load the int value 2 onto the stack |
| iconst\_3 | 06 | 0000 0110 |  | → 3 | load the int value 3 onto the stack |
| iconst\_4 | 07 | 0000 0111 |  | → 4 | load the int value 4 onto the stack |
| iconst\_5 | 08 | 0000 1000 |  | → 5 | load the int value 5 onto the stack |
| idiv | 6c | 0110 1100 |  | value1, value2 → result | divide two integers |
| if\_acmpeq | a5 | 1010 0101 | 2: branchbyte1, branchbyte2 | value1, value2 → | if references are equal, branch to instruction at *branchoffset* (signed short constructed from unsigned bytes branchbyte1 << 8 + branchbyte2) |
| if\_acmpne | a6 | 1010 0110 | 2: branchbyte1, branchbyte2 | value1, value2 → | if references are not equal, branch to instruction at *branchoffset* (signed short constructed from unsigned bytes branchbyte1 << 8 + branchbyte2) |
| if\_icmpeq | 9f | 1001 1111 | 2: branchbyte1, branchbyte2 | value1, value2 → | if ints are equal, branch to instruction at *branchoffset* (signed short constructed from unsigned bytes branchbyte1 << 8 + branchbyte2) |
| if\_icmpge | a2 | 1010 0010 | 2: branchbyte1, branchbyte2 | value1, value2 → | if *value1* is greater than or equal to *value2*, branch to instruction at *branchoffset* (signed short constructed from unsigned bytes branchbyte1 << 8 + branchbyte2) |
| if\_icmpgt | a3 | 1010 0011 | 2: branchbyte1, branchbyte2 | value1, value2 → | if *value1* is greater than *value2*, branch to instruction at *branchoffset* (signed short constructed from unsigned bytes branchbyte1 << 8 + branchbyte2) |
| if\_icmple | a4 | 1010 0100 | 2: branchbyte1, branchbyte2 | value1, value2 → | if *value1* is less than or equal to *value2*, branch to instruction at *branchoffset* (signed short constructed from unsigned bytes branchbyte1 << 8 + branchbyte2) |
| if\_icmplt | a1 | 1010 0001 | 2: branchbyte1, branchbyte2 | value1, value2 → | if *value1* is less than *value2*, branch to instruction at *branchoffset* (signed short constructed from unsigned bytes branchbyte1 << 8 + branchbyte2) |
| if\_icmpne | a0 | 1010 0000 | 2: branchbyte1, branchbyte2 | value1, value2 → | if ints are not equal, branch to instruction at *branchoffset* (signed short constructed from unsigned bytes branchbyte1 << 8 + branchbyte2) |
| ifeq | 99 | 1001 1001 | 2: branchbyte1, branchbyte2 | value → | if *value* is 0, branch to instruction at *branchoffset* (signed short constructed from unsigned bytes branchbyte1 << 8 + branchbyte2) |
| ifge | 9c | 1001 1100 | 2: branchbyte1, branchbyte2 | value → | if *value* is greater than or equal to 0, branch to instruction at *branchoffset* (signed short constructed from unsigned bytes branchbyte1 << 8 + branchbyte2) |
| ifgt | 9d | 1001 1101 | 2: branchbyte1, branchbyte2 | value → | if *value* is greater than 0, branch to instruction at *branchoffset* (signed short constructed from unsigned bytes branchbyte1 << 8 + branchbyte2) |
| ifle | 9e | 1001 1110 | 2: branchbyte1, branchbyte2 | value → | if *value* is less than or equal to 0, branch to instruction at *branchoffset* (signed short constructed from unsigned bytes branchbyte1 << 8 + branchbyte2) |
| iflt | 9b | 1001 1011 | 2: branchbyte1, branchbyte2 | value → | if *value* is less than 0, branch to instruction at *branchoffset* (signed short constructed from unsigned bytes branchbyte1 << 8 + branchbyte2) |
| ifne | 9a | 1001 1010 | 2: branchbyte1, branchbyte2 | value → | if *value* is not 0, branch to instruction at *branchoffset* (signed short constructed from unsigned bytes branchbyte1 << 8 + branchbyte2) |
| ifnonnull | c7 | 1100 0111 | 2: branchbyte1, branchbyte2 | value → | if *value* is not null, branch to instruction at *branchoffset* (signed short constructed from unsigned bytes branchbyte1 << 8 + branchbyte2) |
| ifnull | c6 | 1100 0110 | 2: branchbyte1, branchbyte2 | value → | if *value* is null, branch to instruction at *branchoffset* (signed short constructed from unsigned bytes branchbyte1 << 8 + branchbyte2) |
| iinc | 84 | 1000 0100 | 2: index, const | [No change] | increment local variable *#index* by signed byte *const* |
| iload | 15 | 0001 0101 | 1: index | → value | load an int *value* from a local variable *#index* |
| iload\_0 | 1a | 0001 1010 |  | → value | load an int *value* from local variable 0 |
| iload\_1 | 1b | 0001 1011 |  | → value | load an int *value* from local variable 1 |
| iload\_2 | 1c | 0001 1100 |  | → value | load an int *value* from local variable 2 |
| iload\_3 | 1d | 0001 1101 |  | → value | load an int *value* from local variable 3 |
| impdep1 | fe | 1111 1110 |  |  | reserved for implementation-dependent operations within debuggers; should not appear in any class file |
| impdep2 | ff | 1111 1111 |  |  | reserved for implementation-dependent operations within debuggers; should not appear in any class file |
| imul | 68 | 0110 1000 |  | value1, value2 → result | multiply two integers |
| ineg | 74 | 0111 0100 |  | value → result | negate int |
| instanceof | c1 | 1100 0001 | 2: indexbyte1, indexbyte2 | objectref → result | determines if an object *objectref* is of a given type, identified by class reference *index* in constant pool (indexbyte1 << 8 + indexbyte2) |
| invokedynamic | ba | 1011 1010 | 4: indexbyte1, indexbyte2, 0, 0 | [arg1, [arg2 ...]] → result | invokes a dynamic method and puts the result on the stack (might be void); the method is identified by method reference *index* in constant pool (indexbyte1 << 8 + indexbyte2) |
| invokeinterface | b9 | 1011 1001 | 4: indexbyte1, indexbyte2, count, 0 | objectref, [arg1, arg2, ...] → result | invokes an interface method on object *objectref* and puts the result on the stack (might be void); the interface method is identified by method reference *index* in constant pool (indexbyte1 << 8 + indexbyte2) |
| invokespecial | b7 | 1011 0111 | 2: indexbyte1, indexbyte2 | objectref, [arg1, arg2, ...] → result | invoke instance method on object *objectref* and puts the result on the stack (might be void); the method is identified by method reference *index* in constant pool (indexbyte1 << 8 + indexbyte2) |
| invokestatic | b8 | 1011 1000 | 2: indexbyte1, indexbyte2 | [arg1, arg2, ...] → result | invoke a static method and puts the result on the stack (might be void); the method is identified by method reference *index* in constant pool (indexbyte1 << 8 + indexbyte2) |
| invokevirtual | b6 | 1011 0110 | 2: indexbyte1, indexbyte2 | objectref, [arg1, arg2, ...] → result | invoke virtual method on object *objectref* and puts the result on the stack (might be void); the method is identified by method reference *index* in constant pool (indexbyte1 << 8 + indexbyte2) |
| ior | 80 | 1000 0000 |  | value1, value2 → result | bitwise int OR |
| irem | 70 | 0111 0000 |  | value1, value2 → result | logical int remainder |
| ireturn | ac | 1010 1100 |  | value → [empty] | return an integer from a method |
| ishl | 78 | 0111 1000 |  | value1, value2 → result | int shift left |
| ishr | 7a | 0111 1010 |  | value1, value2 → result | int arithmetic shift right |
| istore | 36 | 0011 0110 | 1: index | value → | store int *value* into variable *#index* |
| istore\_0 | 3b | 0011 1011 |  | value → | store int *value* into variable 0 |
| istore\_1 | 3c | 0011 1100 |  | value → | store int *value* into variable 1 |
| istore\_2 | 3d | 0011 1101 |  | value → | store int *value* into variable 2 |
| istore\_3 | 3e | 0011 1110 |  | value → | store int *value* into variable 3 |
| isub | 64 | 0110 0100 |  | value1, value2 → result | int subtract |
| iushr | 7c | 0111 1100 |  | value1, value2 → result | int logical shift right |
| ixor | 82 | 1000 0010 |  | value1, value2 → result | int xor |
| jsr | a8 | 1010 1000 | 2: branchbyte1, branchbyte2 | → address | jump to subroutine at *branchoffset* (signed short constructed from unsigned bytes branchbyte1 << 8 + branchbyte2) and place the return address on the stack |
| jsr\_w | c9 | 1100 1001 | 4: branchbyte1, branchbyte2, branchbyte3, branchbyte4 | → address | jump to subroutine at *branchoffset* (signed int constructed from unsigned bytes branchbyte1 << 24 + branchbyte2 << 16 + branchbyte3 << 8 + branchbyte4) and place the return address on the stack |
| l2d | 8a | 1000 1010 |  | value → result | convert a long to a double |
| l2f | 89 | 1000 1001 |  | value → result | convert a long to a float |
| l2i | 88 | 1000 1000 |  | value → result | convert a long to a int |
| ladd | 61 | 0110 0001 |  | value1, value2 → result | add two longs |
| laload | 2f | 0010 1111 |  | arrayref, index → value | load a long from an array |
| land | 7f | 0111 1111 |  | value1, value2 → result | [bitwise](https://en.wikipedia.org/wiki/Bitwise_operation) AND of two longs |
| lastore | 50 | 0101 0000 |  | arrayref, index, value → | store a long to an array |
| lcmp | 94 | 1001 0100 |  | value1, value2 → result | push 0 if the two longs are the same, 1 if value1 is greater than value2, -1 otherwise |
| lconst\_0 | 09 | 0000 1001 |  | → 0L | push *0L* (the number [zero](https://en.wikipedia.org/wiki/Zero) with type *long*) onto the stack |
| lconst\_1 | 0a | 0000 1010 |  | → 1L | push *1L* (the number [one](https://en.wikipedia.org/wiki/One) with type *long*) onto the stack |
| ldc | 12 | 0001 0010 | 1: index | → value | push a constant *#index* from a constant pool (String, int, float, Class, java.lang.invoke.MethodType, java.lang.invoke.MethodHandle, or a dynamically-computed constant) onto the stack |
| ldc\_w | 13 | 0001 0011 | 2: indexbyte1, indexbyte2 | → value | push a constant *#index* from a constant pool (String, int, float, Class, java.lang.invoke.MethodType, java.lang.invoke.MethodHandle, or a dynamically-computed constant) onto the stack (wide *index* is constructed as indexbyte1 << 8 + indexbyte2) |
| ldc2\_w | 14 | 0001 0100 | 2: indexbyte1, indexbyte2 | → value | push a constant *#index* from a constant pool (double, long, or a dynamically-computed constant) onto the stack (wide *index* is constructed as indexbyte1 << 8 + indexbyte2) |
| ldiv | 6d | 0110 1101 |  | value1, value2 → result | divide two longs |
| lload | 16 | 0001 0110 | 1: index | → value | load a long value from a local variable *#index* |
| lload\_0 | 1e | 0001 1110 |  | → value | load a long value from a local variable 0 |
| lload\_1 | 1f | 0001 1111 |  | → value | load a long value from a local variable 1 |
| lload\_2 | 20 | 0010 0000 |  | → value | load a long value from a local variable 2 |
| lload\_3 | 21 | 0010 0001 |  | → value | load a long value from a local variable 3 |
| lmul | 69 | 0110 1001 |  | value1, value2 → result | multiply two longs |
| lneg | 75 | 0111 0101 |  | value → result | negate a long |
| lookupswitch | ab | 1010 1011 | 8+: <0–3 bytes padding>, defaultbyte1, defaultbyte2, defaultbyte3, defaultbyte4, npairs1, npairs2, npairs3, npairs4, match-offset pairs... | key → | a target address is looked up from a table using a key and execution continues from the instruction at that address |
| lor | 81 | 1000 0001 |  | value1, value2 → result | bitwise OR of two longs |
| lrem | 71 | 0111 0001 |  | value1, value2 → result | remainder of division of two longs |
| lreturn | ad | 1010 1101 |  | value → [empty] | return a long value |
| lshl | 79 | 0111 1001 |  | value1, value2 → result | bitwise shift left of a long *value1* by int *value2* positions |
| lshr | 7b | 0111 1011 |  | value1, value2 → result | bitwise shift right of a long *value1* by int *value2* positions |
| lstore | 37 | 0011 0111 | 1: index | value → | store a long *value* in a local variable *#index* |
| lstore\_0 | 3f | 0011 1111 |  | value → | store a long *value* in a local variable 0 |
| lstore\_1 | 40 | 0100 0000 |  | value → | store a long *value* in a local variable 1 |
| lstore\_2 | 41 | 0100 0001 |  | value → | store a long *value* in a local variable 2 |
| lstore\_3 | 42 | 0100 0010 |  | value → | store a long *value* in a local variable 3 |
| lsub | 65 | 0110 0101 |  | value1, value2 → result | subtract two longs |
| lushr | 7d | 0111 1101 |  | value1, value2 → result | bitwise shift right of a long *value1* by int *value2* positions, unsigned |
| lxor | 83 | 1000 0011 |  | value1, value2 → result | bitwise XOR of two longs |
| monitorenter | c2 | 1100 0010 |  | objectref → | enter monitor for object ("grab the lock" – start of synchronized() section) |
| monitorexit | c3 | 1100 0011 |  | objectref → | exit monitor for object ("release the lock" – end of synchronized() section) |
| multianewarray | c5 | 1100 0101 | 3: indexbyte1, indexbyte2, dimensions | count1, [count2,...] → arrayref | create a new array of *dimensions* dimensions of type identified by class reference in constant pool *index* (indexbyte1 << 8 + indexbyte2); the sizes of each dimension is identified by *count1*, [*count2*, etc.] |
| new | bb | 1011 1011 | 2: indexbyte1, indexbyte2 | → objectref | create new object of type identified by class reference in constant pool *index* (indexbyte1 << 8 + indexbyte2) |
| newarray | bc | 1011 1100 | 1: atype | count → arrayref | create new array with *count* elements of primitive type identified by *atype* |
| nop | 00 | 0000 0000 |  | [No change] | perform no operation |
| pop | 57 | 0101 0111 |  | value → | discard the top value on the stack |
| pop2 | 58 | 0101 1000 |  | {value2, value1} → | discard the top two values on the stack (or one value, if it is a double or long) |
| putfield | b5 | 1011 0101 | 2: indexbyte1, indexbyte2 | objectref, value → | set field to *value* in an object *objectref*, where the field is identified by a field reference *index* in constant pool (indexbyte1 << 8 + indexbyte2) |
| putstatic | b3 | 1011 0011 | 2: indexbyte1, indexbyte2 | value → | set static field to *value* in a class, where the field is identified by a field reference *index* in constant pool (indexbyte1 << 8 + indexbyte2) |
| ret | a9 | 1010 1001 | 1: index | [No change] | continue execution from address taken from a local variable *#index* (the asymmetry with jsr is intentional) |
| return | b1 | 1011 0001 |  | → [empty] | return void from method |
| saload | 35 | 0011 0101 |  | arrayref, index → value | load short from array |
| sastore | 56 | 0101 0110 |  | arrayref, index, value → | store short to array |
| sipush | 11 | 0001 0001 | 2: byte1, byte2 | → value | push a short onto the stack as an integer *value* |
| swap | 5f | 0101 1111 |  | value2, value1 → value1, value2 | swaps two top words on the stack (note that value1 and value2 must not be double or long) |
| tableswitch | aa | 1010 1010 | 16+: [0–3 bytes padding], defaultbyte1, defaultbyte2, defaultbyte3, defaultbyte4, lowbyte1, lowbyte2, lowbyte3, lowbyte4, highbyte1, highbyte2, highbyte3, highbyte4, jump offsets... | index → | continue execution from an address in the table at offset *index* |
| wide | c4 | 1100 0100 | 3/5: opcode, indexbyte1, indexbyte2 or iinc, indexbyte1, indexbyte2, countbyte1, countbyte2 | [same as for corresponding instructions] | execute *opcode*, where *opcode* is either iload, fload, aload, lload, dload, istore, fstore, astore, lstore, dstore, or ret, but assume the *index* is 16 bit; or execute iinc, where the *index* is 16 bits and the constant to increment by is a signed 16 bit short |
| *(no name)* | cb-fd |  |  |  | these values are currently unassigned for opcodes and are reserved for future use |