

aaload

Operation Load reference from array

Format

aaload

Forms aaload = 50 (0x32)

Operand Stack ..., arrayref, index ⇒ ..., value

aastore

Operation Store into reference array

Format

aastore

Forms aastore = 83 (0x53)

Operand Stack ..., arrayref, index, value ⇒ ...

aconst_null

Operation Push null

Format

aconst_null

Forms aconst_null = 1 (0x1)

Operand Stack ... ⇒ ..., null

aload

Operation Load reference from local variable

Format

aload
index

Forms aload = 25 (0x19)

arraylength

Forms arraylength = 190 (0xbe)

Operand Stack ..., arrayref ⇒ ..., length

astore

Operation Store reference into local variable

Format

astore
index

Forms astore = 58 (0x3a)

Operand Stack ..., objectref ⇒ ...

astore_<n>

Operation Store reference into local variable

Format

astore_<n>

Forms astore_0 = 75 (0x4b) astore_1 = 76 (0x4c) astore_2 = 77 (0x4d) astore_3 = 78 (0x4e)

Operand Stack ..., objectref ⇒ ...

athrow

Operation Throw exception or error

Format

athrow

Forms athrow = 191 (0xbf)

Operand Stack ..., objectref ⇒ objectref

baload

Operation Load byte or boolean from array

Format

baload

Forms baload = 51 (0x33)

Operand Stack ..., arrayref, index ⇒ ..., value

bastore

Operation Store into byte or boolean array

Format

bastore

Forms bastore = 84 (0x54)

Operand Stack ..., arrayref, index, value ⇒ ...

bipush

Operation Push byte

Format

bipush
byte

Forms bipush = 16 (0x10)

Operand Stack ... ⇒ ..., value

caload

Operation Load char from array

Format

caload

Forms caload = 52 (0x34)

Operand Stack ..., arrayref, index ⇒ ..., value

Operand Stack ... ⇒ ..., objectref

aload_<n>

Operation Load reference from local variable

Format

aload_<n>

Forms aload_0 = 42 (0x2a) aload_1 = 43 (0x2b) aload_2 = 44 (0x2c) aload_3 = 45 (0x2d)

Operand Stack ... ⇒ ..., objectref

anewarray

Operation Create new array of reference

Format

anewarray
indexbyte1
indexbyte2

Forms anewarray = 189 (0xbd)

Operand Stack ..., count ⇒ ..., arrayref

areturn

Operation Return reference from method

Format

areturn

Forms areturn = 176 (0xb0)

Operand Stack ..., objectref ⇒ [empty]

arraylength

Operation Get length of array

Format

castore

Operation Store into char array

Format

castore

Forms castore = 85 (0x55)

Operand Stack ..., arrayref, index, value ⇒ ...

checkcast

Operation Check whether object is of given type

Format

checkcast
indexbyte1
indexbyte2

Forms checkcast = 192 (0xc0)

Operand Stack ..., objectref ⇒ ..., objectref

d2f

Operation Convert double to float

Format

d2f

Forms d2f = 144 (0x90)

Operand Stack ..., value ⇒ ..., result

d2i

Operation Convert double to int

Format

d2i

Forms d2i = 142 (0x8e)

Operand Stack ..., value ⇒ ..., result

d2l

Operation Convert double to long

Format

d2l

Forms d2l = 143 (0x8f)

Operand Stack ..., value ⇒ ..., result

dadd

Operation Add double

Format

dadd

Forms dadd = 99 (0x63)

Operand Stack ..., value1, value2 ⇒ ..., result

daload

Operation Load double from array

Format

daload

Forms daload = 49 (0x31)

Operand Stack ..., arrayref, index ⇒ ..., value

dastore

Operation Store into double array

Format

dastore

Forms dastore = 82 (0x52)

Operand Stack ..., arrayref, index, value ⇒ ...

dcmp<op>

Operation Compare double

Format

dcmp<op>

Forms dcmpg = 152 (0x98) dcmpl = 151 (0x97)

Operand Stack ..., value1, value2 ⇒ ..., result

dconst_<d>

Operation Push double

Format

dconst_<d>

Forms dconst_0 = 14 (0xe) dconst_1 = 15 (0xf)

Operand Stack ... ⇒ ..., <d>

ddiv

Operation Divide double

Format

ddiv

Forms ddiv = 111 (0x6f)

Operand Stack ..., value1, value2 ⇒ ..., result

dload

Operation Load double from local variable

Format

dload
index

Forms dload = 24 (0x18)

Operand Stack ... ⇒ ..., value

dload_<n>

Operation Load double from local variable

Format

dload_<n>

Forms dload_0 = 38 (0x26) dload_1 = 39 (0x27) dload_2 = 40 (0x28) dload_3 = 41 (0x29)

Operand Stack ... ⇒ ..., value

dmul

Operation Multiply double

Format

dmul

Forms dmul = 107 (0x6b)

Operand Stack ..., value1, value2 ⇒ ..., result

dneg

Operation Negate double

Format

dneg

Forms dneg = 119 (0x77)

Operand Stack ..., value ⇒ ..., result

drem

Operation Remainder double

Format

drem

Forms *drem* = 115 (0x73)
Operand Stack ..., *value1*, *value2* ⇒ ..., *result*

dreturn

Operation Return double from method
Format

dreturn

Forms *dreturn* = 175 (0xaf)
Operand Stack ..., *value* ⇒ [empty]

dstore

Operation Store double into local variable
Format

dstoreindex

Forms *dstore* = 57 (0x39)
Operand Stack ..., *value* ⇒ ...

dstore_<n>

Operation Store double into local variable
Format

dstore_<n>

Forms *dstore_0* = 71 (0x47) *dstore_1* = 72 (0x48) *dstore_2* = 73 (0x49) *dstore_3* = 74 (0x4a)
Operand Stack ..., *value* ⇒ ...

dsub

Operation Subtract double
Format

dsub

Forms *dsub* = 103 (0x67)
Operand Stack ..., *value1*, *value2* ⇒ ..., *result*

dup

Operation Duplicate the top operand stack value
Format

dup

Forms *dup* = 89 (0x59)
Operand Stack ..., *value* ⇒ ..., *value*, *value*

dup_x1

Operation Duplicate the top operand stack value and insert two values down
Format

dup_x1

Forms *dup_x1* = 90 (0x5a)
Operand Stack ..., *value2*, *value1* ⇒ ..., *value1*, *value2*, *value1*

dup_x2

Operation Duplicate the top operand stack value and insert two or three values down
Format

dup_x2

Forms *dup_x2* = 91 (0x5b)
Operand Stack Form 1:
..., *value3*, *value2*, *value1* ⇒ ..., *value1*, *value3*, *value2*, *value1*
where *value1*, *value2*, and *value3* are all values of a category 1 computational type (§3.11.1).
Form 2:

..., *value2*, *value1* ⇒ ..., *value1*, *value2*, *value1*

where *value1* is a value of a category 1 computational type and *value2* is a value of a category 2 computational type (§3.11.1).

dup2

Operation Duplicate the top one or two operand stack values
Format

dup2

Forms *dup2* = 92 (0x5c)
Operand Stack Form 1:
..., *value2*, *value1* ⇒ ..., *value2*, *value1*, *value2*, *value1*
where both *value1* and *value2* are values of a category 1 computational type (§3.11.1).
Form 2:
..., *value* ⇒ ..., *value*, *value*
where *value* is a value of a category 2 computational type (§3.11.1).

dup2_x1

Operation Duplicate the top one or two operand stack values and insert two or three values down
Format

dup2_x1

Forms *dup2_x1* = 93 (0x5d)
Operand Stack Form 1:
..., *value3*, *value2*, *value1* ⇒ ..., *value2*, *value1*, *value3*, *value2*, *value1*
where *value1*, *value2*, and *value3* are all values of a category 1 computational type (§3.11.1).
Form 2:
..., *value2*, *value1* ⇒ ..., *value1*, *value2*, *value1*
where *value1* is a value of a category 2 computational type and *value2* is a value of a category 1 computational type (§3.11.1).

dup2_x2

Operation Duplicate the top one or two operand stack values and insert two, three, or four values down
Format

dup2_x2

Forms *dup2_x2* = 94 (0x5e)
Operand Stack Form 1:
..., *value4*, *value3*, *value2*, *value1* ⇒ ..., *value2*, *value1*, *value4*, *value3*, *value2*, *value1*
where *value1*, *value2*, *value3*, and *value4* are all values of a category 1 computational type (§3.11.1).
Form 2:
..., *value3*, *value2*, *value1* ⇒ ..., *value1*, *value3*, *value2*, *value1*
where *value1* is a value of a category 2 computational type and *value2* and *value3* are both values of a category 1 computational type (§3.11.1).
Form 3:
..., *value3*, *value2*, *value1* ⇒ ..., *value2*, *value1*, *value3*, *value2*, *value1*
where *value1* and *value2* are both values of a category 1 computational type and *value3* is a value of a category 2 computational type (§3.11.1).
Form 4:
..., *value2*, *value1* ⇒ ..., *value1*, *value2*, *value1*
where *value1* and *value2* are both values of a category 2 computational type (§3.11.1).

f2d

Operation Convert float to double
Format

f2d

Forms *f2d* = 141 (0x8d)
Operand Stack ..., *value* ⇒ ..., *result*

f2i

Operation Convert float to int

Format



Forms f2i = 139 (0x8b)

Operand Stack ..., value ⇒ ..., result

f2l

Operation Convert float to long

Format



Forms f2l = 140 (0x8c)

Operand Stack ..., value ⇒ ..., result

fadd

Operation Add float

Format



Forms fadd = 98 (0x62)

Operand Stack ..., value1, value2 ⇒ ..., result

faload

Operation Load float from array

Format



Forms faload = 48 (0x30)

Operand Stack ..., arrayref, index ⇒ ..., value

fload

Operation Load float from local variable

Format



Forms fload = 23 (0x17)

Operand Stack ...|



fastore

Operation Store into float array

Format



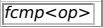
Forms fastore = 81 (0x51)

Operand Stack ..., arrayref, index, value ⇒ ...

fcmp<op>

Operation Compare float

Format



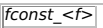
Forms fcmpg = 150 (0x96) fcmpl = 149 (0x95)

Operand Stack ..., value1, value2 ⇒ ..., result

fconst_<f>

Operation Push float

Format



Forms fconst_0 = 11 (0xb) fconst_1 = 12 (0xc) fconst_2 = 13 (0xd)

Operand Stack ... ⇒ ..., <f>

fdiv

Operation Divide float

Format



Forms fddiv = 110 (0x6e)

Operand Stack ..., value1, value2 ⇒ ..., result

Operand Stack

..., value ⇒ ...

fsub

Operation

Subtract float

Format

fsub

Forms

fsub = 102 (0x66)

Operand Stack

..., value1, value2 ⇒ ..., result

getfield

Operation

Fetch field from object

Format

getfield

indexbyte1

indexbyte2

Forms

getfield = 180 (0xb4)

Operand Stack

..., objectref ⇒ ..., value

getstatic

Operation

Get static field from class

Format

getstatic

indexbyte1

indexbyte2

Forms

getstatic = 178 (0xb2)

Operand Stack

..., ⇒ ..., value

goto

Operation

Branch always

Format

goto

branchbyte1

branchbyte2

Forms

goto = 167 (0xa7)

Operand Stack

No change

goto_w

Operation

Branch always (wide index)

Format

goto_w

branchbyte1

branchbyte2

branchbyte3

branchbyte4

Forms

goto_w = 200 (0xc8)

Operand Stack

No change

i2b

Operation

Convert int to byte

Format

i2b

Forms

i2b = 145 (0x91)

Operand Stack

..., value ⇒ ..., result

i2c

Operation

Convert int to char

Format

i2c

Forms

i2c = 146 (0x92)

Operand Stack

..., value ⇒ ..., result

i2d

Operation

Convert int to double

Format

i2d

Forms

i2d = 135 (0x87)

Operand Stack

..., value ⇒ ..., result

i2f

Operation

Convert int to float

Format

i2f

Forms

i2f = 134 (0x86)

Operand Stack

..., value ⇒ ..., result

i2l

Operation

Convert int to long

Format

i2l

Forms

i2l = 133 (0x85)

Operand Stack

..., value ⇒ ..., result

i2s

Operation

Convert int to short

Format

i2s

Forms

i2s = 147 (0x93)

Operand Stack

..., value ⇒ ..., result

iadd

Operation

Add int

Format

iadd

Forms

iadd = 96 (0x60)

Operand Stack

..., value1, value2 ⇒ ..., result

iaload

Operation

Load int from array

Format

iaload

Forms

iaload = 46 (0x2e)

Operand Stack

..., arrayref, index ⇒ ..., value

iand

Operation

Boolean AND int

Format

iand

Forms

iand = 126 (0x7e)

Operand Stack

..., value1, value2 ⇒ ..., result

iastore

Operation

Store into int array

Format

iastore

Forms *iastore* = 79 (0x4f)

Operand Stack ..., arrayref, index, value ⇒ ...

iconst_<i>

Operation Push int constant

Format

iconst_<i>

Forms *iconst_m1* = 2 (0x2) *iconst_0* = 3 (0x3) *iconst_1* = 4 (0x4) *iconst_2* = 5 (0x5) *iconst_3* = 6 (0x6) *iconst_4* = 7 (0x7) *iconst_5* = 8 (0x8)

Operand Stack ... ⇒ ..., <i>

idiv

Operation Divide int

Format

idiv

Forms *idiv* = 108 (0x6c)

Operand Stack ..., value1, value2 ⇒ ..., result

if_acmp<cond>

Operation Branch if reference comparison succeeds

Format

if_acmp<cond>
branchbyte1
branchbyte2

Forms *if_acmpeq* = 165 (0xa5) *if_acmpne* = 166 (0xa6)

Operand Stack ..., value1, value2 ⇒ ...

if_icmp<cond>

Operation Branch if int comparison succeeds

Format

if_icmp<cond>
branchbyte1
branchbyte2

Forms *if_icmpeq* = 159 (0x9f) *if_icmpne* = 160 (0xa0) *if_icmplt* = 161 (0xa1) *if_icmpge* = 162 (0xa2) *if_icmpgt* = 163 (0xa3) *if_icmple* = 164 (0xa4)

Operand Stack ..., value1, value2 ⇒ ...

if<cond>

Operation Branch if int comparison with zero succeeds

Format

if<cond>
branchbyte1
branchbyte2

Forms *ifeq* = 153 (0x99) *ifne* = 154 (0x9a) *iflt* = 155 (0x9b) *ifge* = 156 (0x9c) *ifgt* = 157 (0x9d) *ifle* = 158 (0x9e)

Operand Stack ..., value ⇒ ...

ifnonnull

Operation Branch if reference not null

Format

ifnonnull
branchbyte1
branchbyte2

Forms *ifnonnull* = 199 (0xc7)

Operand Stack ..., value ⇒ ...

ifnull

Operation Branch if reference is null

Format

ifnull

branchbyte1
branchbyte2

Forms *ifnull* = 198 (0xc6)

Operand Stack ..., value ⇒ ...

iinc

Operation Increment local variable by constant

Format

iinc
index
const

Forms *iinc* = 132 (0x84)

Operand Stack No change

iload

Operation Load int from local variable

Format

iload
index

Forms *iload* = 21 (0x15)

Operand Stack ... ⇒ ..., value

iload_<n>

Operation Load int from local variable

Format

iload_<n>

Forms *iload_0* = 26 (0x1a) *iload_1* = 27 (0x1b) *iload_2* = 28 (0x1c) *iload_3* = 29 (0x1d)

Operand Stack ... ⇒ ..., value

imul

Operation Multiply int

Format

imul

Forms *imul* = 104 (0x68)

Operand Stack ..., value1, value2 ⇒ ..., result

ineg

Operation Negate int

Format

ineg

Forms *ineg* = 116 (0x74)

Operand Stack ..., value ⇒ ..., result

instanceof

Operation

Determine if object is of given type

Format

instanceof
indexbyte1
indexbyte2

Forms *instanceof* = 193 (0xc1)

Operand Stack ..., objectref ⇒ ..., result

invokeinterface

Operation Invoke interface method

Format

invokeinterface
indexbyte1

<i>indexbyte2</i>
<i>count</i>
<i>0</i>

Forms *invokeinterface* = 185 (0xb9)

Operand Stack ..., *objectref*, [*arg1*, [*arg2* ...]] ⇒ ...

invokespecial

Operation Invoke instance method; special handling for superclass, private, and instance initialization method invocations

Format

<i>invokespecial</i>
<i>indexbyte1</i>
<i>indexbyte2</i>

Forms *invokespecial* = 183 (0xb7)

Operand Stack ..., *objectref*, [*arg1*, [*arg2* ...]] ⇒ ...

invokestatic

Operation Invoke a class (static) method

Format

<i>invokestatic</i>
<i>indexbyte1</i>
<i>indexbyte2</i>

Forms *invokestatic* = 184 (0xb8)

Operand Stack ..., [*arg1*, [*arg2* ...]] ⇒ ...

invokevirtual

Operation Invoke instance method; dispatch based on class

Format

<i>invokevirtual</i>
<i>indexbyte1</i>
<i>indexbyte2</i>

Forms *invokevirtual* = 182 (0xb6)

Operand Stack ..., *objectref*, [*arg1*, [*arg2* ...]] ⇒ ...

ior

Operation Boolean OR int

Format

<i>ior</i>

Forms *ior* = 128 (0x80)

Operand Stack ..., *value1*, *value2* ⇒ ..., *result*

irem

Operation Remainder int

Format

<i>irem</i>

Forms *irem* = 112 (0x70)

Operand Stack ..., *value1*, *value2* ⇒ ..., *result*

ireturn

Operation Return int from method

Format

<i>ireturn</i>

Forms *ireturn* = 172 (0xac)

Operand Stack ..., *value* ⇒ [empty]

ishl

Operation Shift left int

Format

<i>ishl</i>

Forms *ishl* = 120 (0x78)

Operand Stack ..., *value1*, *value2* ⇒ ..., *result*

ishr

Operation Arithmetic shift right int

Format

<i>ishr</i>

Forms *ishr* = 122 (0x7a)

Operand Stack ..., *value1*, *value2* ⇒ ..., *result*

istore

Operation Store int into local variable

Format

<i>istore</i>
<i>index</i>

Forms *istore* = 54 (0x36)

Operand Stack ..., *value* ⇒ ...

istore_<n>

Operation Store int into local variable

Format

<i>istore_<n></i>

Forms *istore_0* = 59 (0x3b) *istore_1* = 60 (0x3c) *istore_2* = 61 (0x3d) *istore_3* = 62 (0x3e)

Operand Stack ..., *value* ⇒ ...

isub

Operation Subtract int

Format

<i>isub</i>

Forms *isub* = 100 (0x64)

Operand Stack ..., *value1*, *value2* ⇒ ..., *result*

iushr

Operation Logical shift right int

Format

<i>iushr</i>

Forms *iushr* = 124 (0x7c)

Operand Stack ..., *value1*, *value2* ⇒ ..., *result*

ixor

Operation Boolean XOR int

Format

<i>ixor</i>

Forms *ixor* = 130 (0x82)

Operand Stack ..., *value1*, *value2* ⇒ ..., *result*

jsr

Operation Jump subroutine

Format

<i>jsr</i>
<i>branchbyte1</i>
<i>branchbyte2</i>

Forms *jsr* = 168 (0xa8)

Operand Stack ... ⇒ ..., *address*

jsr_w

Operation Jump subroutine (wide index)

Format

<i>jsr_w</i>
<i>branchbyte1</i>
<i>branchbyte2</i>
<i>branchbyte3</i>
<i>branchbyte4</i>

Forms *jsr_w* = 201 (0xc9)

Operand Stack ... ⇒ ..., *address*

I2d

Operation Convert long to double

Format

<i>I2d</i>

Forms *I2d* = 138 (0x8a)

Operand Stack ..., *value* ⇒ ..., *result*

I2f

Operation Convert long to float

Format

<i>I2f</i>

Forms *I2f* = 137 (0x89)

Operand Stack ..., *value* ⇒ ..., *result*

I2i

Operation Convert long to int

Format

<i>I2i</i>

Forms *I2i* = 136 (0x88)

Operand Stack ..., *value* ⇒ ..., *result*

ladd

Operation Add long

Format

<i>ladd</i>

Forms *ladd* = 97 (0x61)

Operand Stack ..., *value1*, *value2* ⇒ ..., *result*

laload

Operation Load long from array

Format

<i>laload</i>

Forms *laload* = 47 (0x2f)

Operand Stack ..., *arrayref*, *index* ⇒ ..., *value*

land

Operation Boolean AND long

Format

<i>land</i>

Forms *land* = 127 (0x7f)

Operand Stack ..., *value1*, *value2* ⇒ ..., *result*

lastore

Operation Store into long array

Format

<i>lastore</i>

Forms *lastore* = 80 (0x50)

Operand Stack ..., *arrayref*, *index*, *value* ⇒ ...

lcmp

Operation Compare long

Format

<i>lcmp</i>

Forms *lcmp* = 148 (0x94)

Operand Stack ..., *value1*, *value2* ⇒ ..., *result*

lconst_<I>

Operation Push long constant

Format

<i>lconst_<I></i>

Forms *lconst_0* = 9 (0x9) *lconst_1* = 10 (0xa)

Operand Stack ... ⇒ ..., <*I*>

ldc

Operation Push item from runtime constant pool

Format

<i>ldc</i>
<i>index</i>

Forms *ldc* = 18 (0x12)

Operand Stack ... ⇒ ..., *value*

ldc_w

Operation Push item from runtime constant pool (wide index)

Format

<i>ldc_w</i>
<i>indexbyte1</i>

<i>indexbyte2</i>

Forms *ldc_w* = 19 (0x13)

Operand Stack ... ⇒ ..., *value*

ldc2_w

Operation Push long or double from runtime constant pool (wide index)

Format

<i>ldc2_w</i>
<i>indexbyte1</i>
<i>indexbyte2</i>

Forms *ldc2_w* = 20 (0x14)

Operand Stack ... ⇒ ..., *value*

ldiv

Operation Divide long

Format

<i>ldiv</i>

Forms *ldiv* = 109 (0x6d)

Operand Stack ..., *value1*, *value2* ⇒ ..., *result*

lload

Operation Load long from local variable

Format

<i>lload</i>
<i>index</i>

Forms *lload* = 22 (0x16)

Operand Stack ... ⇒ ..., *value*

lload_<n>

Operation Load long from local variable

Format

`lload_<n>`

Forms *lload_0* = 30 (0x1e) *lload_1* = 31 (0x1f) *lload_2* = 32 (0x20) *lload_3* = 33 (0x21)

Operand Stack ... ⇒ ..., *value*

Imul

Operation Multiply long

Format

`lmul`

Forms *lmul* = 105 (0x69)

Operand Stack ..., *value1*, *value2* ⇒ ..., *result*

Ineg

Operation Negate long

Format

`lneg`

Forms *lneg* = 117 (0x75)

Operand Stack ..., *value* ⇒ ..., *result*

lookupswitch

Operation Access jump table by key match and jump

Format

<code>lookupswitch</code>
<code><0-3 byte pad></code>
<code>defaultbyte1</code>
<code>defaultbyte2</code>
<code>defaultbyte3</code>
<code>defaultbyte4</code>

<code>npairs1</code>
<code>npairs2</code>
<code>npairs3</code>
<code>npairs4</code>
<code>match-offset pairs...</code>

Forms *lookupswitch* = 171 (0xab)

Operand Stack ..., *key* ⇒ ...

lor

Operation Boolean OR long

Format

`lor`

Forms *lor* = 129 (0x81)

Operand Stack ..., *value1*, *value2* ⇒ ..., *result*

Irem

Operation Remainder long

Format

`lrem`

Forms *lrem* = 113 (0x71)

Operand Stack ..., *value1*, *value2* ⇒ ..., *result*

Ireturn

Operation Return long from method

Format

`lreturn`

Forms *lreturn* = 173 (0xad)

Operand Stack ..., *value* ⇒ [empty]

lshl

Operation Shift left

Format

`lshl`
Long

Forms *lshl* = 121 (0x79)

Operand Stack ..., *value1*, *value2* ⇒ ..., *result*

lshr

Operation Arithmetic shift right long

Format

`lshr`

Forms *lshr* = 123 (0x7b)

Operand Stack ..., *value1*, *value2* ⇒ ..., *result*

Istore

Operation Store long into local variable

Format

`lstore`
`index`

Forms *lstore* = 55 (0x37)

Operand Stack ..., *value* ⇒ ...

Istore_<n>

Operation Store long into local variable

Format

`lstore_<n>`

Forms *lstore_0* = 63 (0x3f) *lstore_1* = 64 (0x40) *lstore_2* = 65 (0x41) *lstore_3* = 66 (0x42)

Operand Stack ..., *value* ⇒ ...

lsub

Operation Subtract long

Format

`lsub`

Forms *lsub* = 101 (0x65)

Operand Stack ..., *value1*, *value2* ⇒ ..., *result*

lushr

Operation Logical shift right long

Format

`lushr`

Forms *lushr* = 125 (0x7d)

Operand Stack ..., *value1*, *value2* ⇒ ..., *result*

lxor

Operation Boolean XOR long

Format

`lxor`

Forms *lxor* = 131 (0x83)

Operand Stack ..., *value1*, *value2* ⇒ ..., *result*

monitorenter

Operation Enter monitor for object

Format

`monitorenter`

Forms *monitorenter* = 194 (0xc2)

Operand Stack ..., *objectref* ⇒ ...

monitorexit

Operation Exit monitor for object

Format

monitorexit

Forms *monitorexit* = 195 (0xc3)

Operand Stack ..., *objectref* ⇒ ...

multianewarray

Operation Create new multidimensional array

Format

multianewarray
indexbyte1
indexbyte2
dimensions

Forms *multianewarray* = 197 (0xc5)

Operand Stack ..., *count1*, [*count2*, ...] ⇒ ..., *arrayref*

new

Operation Create new object

Format

new
indexbyte1
indexbyte2

Forms *new* = 187 (0xbb)

Operand Stack ... ⇒ ..., *objectref*

newarray

Operation Create new array

Format

newarray
atype

Forms *newarray* = 188 (0xbc)

Operand Stack ..., *count* ⇒ ..., *arrayref*

nop

Operation Do nothing

Format

nop

Forms *nop* = 0 (0x0)

Operand Stack No change

pop

Operation Pop the top operand stack value

Format

pop

Forms *pop* = 87 (0x57)

Operand Stack ..., *value* ⇒ ...

pop2

Operation Pop the top one or two operand stack values

Format

pop2

Forms *pop2* = 88 (0x58)

Operand Stack Form 1:
..., *value2*, *value1* ⇒ ...

where each of *value1* and *value2* is a value of a category 1 computational type ([§3.11.1](#)).

Form 2:

..., *value* ⇒ ...

where *value* is a value of a category 2 computational type ([§3.11.1](#)).

putfield

Operation Set field in object

Format

putfield
indexbyte1
indexbyte2

Forms *putfield* = 181 (0xb5)

Operand Stack ..., *objectref*, *value* ⇒ ...

putstatic

Operation Set static field in class

Format

putstatic
indexbyte1
indexbyte2

Forms *putstatic* = 179 (0xb3)

Operand Stack ..., *value* ⇒ ...

ret

Operation Return from subroutine

Format

ret
index

Forms *ret* = 169 (0xa9)

Operand Stack No change

return

Operation Return void from method

Format

return

Forms *return* = 177 (0xb1)

Operand Stack ... ⇒ [empty]

saload

Operation Load short from array

Format

saload

Forms *saload* = 53 (0x35)

Operand Stack ..., *arrayref*, *index* ⇒ ..., *value*

sastore

Operation Store into short array

Format

sastore

Forms *sastore* = 86 (0x56)

Operand Stack ..., *array*, *index*, *value* ⇒ ...

sipush

Operation Push short

Format

sipush
byte1

byte2

Forms *si*push = 17 (0x11)

Operand Stack ... ⇒ ..., *value*

swap

Operation Swap the top two operand stack values

Format

swap

Forms swap = 95 (0x5f)

Operand Stack ..., *value2*, *value1* ⇒ ..., *value1*, *value2*

tableswitch

Operation Access jump table by index and jump

Format

tableswitch
<0-3 byte pad\>
defaultbyte1
defaultbyte2
defaultbyte3
defaultbyte4
lowbyte1
lowbyte2
lowbyte3
lowbyte4
highbyte1
highbyte2
highbyte3
highbyte4
jump offsets...

Forms tableswitch = 170 (0xaa)

Operand Stack ..., *index* ⇒ ...

wide

Operation Extend local variable index by additional bytes

Format 1

wide
<opcode>
indexbyte1
indexbyte2

where <opcode> is one of *iload*, *fload*, *aload*, *lload*, *dload*, *istore*, *fstore*, *astore*, *lstore*, *dstore*, or *ret*

Format 2

wide
<i>linc</i>
indexbyte1
indexbyte2
constbyte1
constbyte2

Forms wide = 196 (0xc4)

Operand Stack Same as modified instruction