Dalvik VM Instruction Formats

Introduction and Overview

This document lists the instruction formats used by Dalvik bytecode and is meant to be used in conjunction with the bytecode reference document.

Bitwise descriptions

The first column in the format table lists the bitwise layout of the format. It consists of one or more space-separated "words" each of which describes a 16-bit code unit. Each character in a word represents four bits, read from high bits to low, with vertical bars ("`|`") interspersed to aid in reading. Uppercase letters in sequence from "`A`" are used to indicate fields within the format (which then get defined further by the syntax column). The term "`op`" is used to indicate the position of the eight-bit opcode within the format. A slashed zero ("`Ø`") is used to indicate that all bits should be zero in the indicated position.

For example, the format "`B|A|`*`op`*` CCCC`" indicates that the format consists of two 16-bit code units. The first word consists of the opcode in the low eight bits and a pair of four-bit values in the high eight bits; and the second word consists of a single 16-bit value.

Format IDs

The second column in the format table indicates the short identifier for the format, which is used in other documents and in code to identify the format.

Format IDs consist of three characters, two digits followed by a letter. The first digit indicates the number of 16-bit code units in the format. The second digit indicates the maximum number of registers that the format contains (maximum, since some formats can accomodate a variable number of registers), with the special designation "`r`" indicating that a range of registers is encoded. The final letter semi-mnemonically indicates the type of any extra data encoded by the format. For example, format "`21t`" is of length two, contains one register reference, and additionally contains a branch target.

Suggested static linking formats have an additional "`s`" suffix, making them four characters total.

The full list of typecode letters are as follows. Note that some forms have different sizes, depending on the format:

Mnemonic	Bit Sizes	Meaning
b	8	immediate signed **b**yte
c	16, 32	**c**onstant pool index
f	16	inter**f**ace constants (only used in statically linked formats)

h	16	immediate signed **h**at (high-order bits of a 32- or 64-bit value; low-order bits are all 0)
i	32	immediate signed **i**nt, or 32-bit float
l	64	immediate signed **l**ong, or 64-bit double
m	16	**m**ethod constants (only used in statically linked formats)
n	4	immediate signed **n**ibble
s	16	immediate signed **s**hort
t	8, 16, 32	branch **t**arget
x	0	no additional data

Syntax

The third column of the format table indicates the human-oriented syntax for instructions which use the indicated format. Each instruction starts with the named opcode and is optionally followed by one or more arguments, themselves separated with commas.

Wherever an argument refers to a field from the first column, the letter for that field is indicated in the syntax, repeated once for each four bits of the field. For example, an eight-bit field labeled "`BB`" in the first column would also be labeled "`BB`" in the syntax column.

Arguments which name a register have the form "`v`*`X`*". The prefix "`v`" was chosen instead of the more common "`r`" exactly to avoid conflicting with (non-virtual) architectures on which a Dalvik virtual machine might be implemented which themselves use the prefix "`r`" for their registers. (That is, this decision makes it possible to talk about both virtual and real registers together without the need for circumlocution.)

Arguments which indicate a literal value have the form "`#+`*`X`*". Some formats indicate literals that only have non-zero bits in their high-order bits; for these, the zeroes are represented explicitly in the syntax, even though they do not appear in the bitwise representation.

Arguments which indicate a relative instruction address offset have the form "`+`*`X`*".

Arguments which indicate a literal constant pool index have the form "*`kind`*`@`*`X`*", where "*`kind`*" indicates which constant pool is being referred to. Each opcode that uses such a format explicitly allows only one kind of constant; see the opcode reference to figure out the correspondence. The four kinds of constant pool are "`string`" (string pool index), "`type`" (type pool index), "`field`" (field pool index), and "`meth`" (method pool index).

Similar to the representation of constant pool indices, there are also suggested (optional) forms that indicate prelinked offsets or indices. These prelinked values include "`vtaboff`" (vtable offset), "`fieldoff`" (field offset), and "`iface`" (interface pool index).

In the cases where a format value isn't explictly part of the syntax but instead picks a variant, each variant is listed with the prefix "`[`*`X`*`=`*`N`*`]`" (e.g., "`[B=2]`") to indicate the correspondence.

The Formats

Format	ID	Syntax	Notable Opcodes Covered
ØØ|*op*	10x	*op*	
B|A|*op*	12x	*op* vA, vB	
	11n	*op* vA, #+B	
AA|*op*	11x	*op* vAA	
	10t	*op* +AA	goto
ØØ|*op* AAAA	20t	*op* +AAAA	goto/16
AA|*op* BBBB	22x	*op* vAA, vBBBB	
	21t	*op* vAA, +BBBB	
	21s	*op* vAA, #+BBBB	
	21h	*op* vAA, #+BBBB0000 *op* vAA, #+BBBB000000000000	
	21c	*op* vAA, type@BBBB *op* vAA, field@BBBB *op* vAA, string@BBBB	check-cast const-class const-string
AA|*op* CC|BB	23x	*op* vAA, vBB, vCC	
	22b	*op* vAA, vBB, #+CC	
B|A|*op* CCCC	22t	*op* vA, vB, +CCCC	
	22s	*op* vA, vB, #+CCCC	
	22c	*op* vA, vB, type@CCCC *op* vA, vB, field@CCCC	instance-of
	22cs	*op* vA, vB, fieldoff@CCCC	*(suggested format for statically linked field access instructions of format 22c)*
ØØ|*op* AAAA_{lo} AAAA_{hi}	30t	*op* +AAAAAAAA	goto/32
ØØ|*op* AAAA BBBB	32x	*op* vAAAA, vBBBB	
AA|*op* BBBB_{lo} BBBB_{hi}	31i	*op* vAA, #+BBBBBBBB	
	31t	*op* vAA, +BBBBBBBB	
	31c	*op* vAA, string@BBBBBBBB	const-string/jumbo
B|A|*op* CCCC G|F|E|D	35c	*[B=5] op* {vD, vE, vF, vG, vA}, meth@CCCC *[B=5] op* {vD, vE, vF, vG, vA}, type@CCCC *[B=4] op* {vD, vE, vF, vG}, *kind*@CCCC *[B=3] op* {vD, vE, vF}, *kind*@CCCC *[B=2] op* {vD, vE}, *kind*@CCCC *[B=1] op* {vD}, *kind*@CCCC *[B=0] op* {}, *kind*@CCCC	
B|A|*op* CCCC G|F|E|D	35ms	*[B=5] op* {vD, vE, vF, vG, vA}, vtaboff@CCCC *[B=4] op* {vD, vE, vF, vG}, vtaboff@CCCC *[B=3] op* {vD, vE, vF}, vtaboff@CCCC *[B=2] op* {vD, vE}, vtaboff@CCCC *[B=1] op* {vD}, vtaboff@CCCC	*(suggested format for statically linked* invoke-virtual *and* invoke-super *instructions of format 35c)*
		[B=5] op vB, {vE, vF, vG, vH, vA}, vtaboff@CC, iface@DD	

B|A|*op* DDCC H|G|F|E	35fs	vtaboff@CC, iface@DD *[B=4]* *op* vB, {vE, vF, vG, vH}, vtaboff@CC, iface@DD *[B=3]* *op* vB, {vE, vF, vG}, vtaboff@CC, iface@DD *[B=2]* *op* vB, {vE, vF}, vtaboff@CC, iface@DD *[B=1]* *op* vB, {vE}, vtaboff@CC, iface@DD	*(suggested format for statically linked invoke-interface instructions of format 35c)*
AA|*op* BBBB CCCC	3rc	*op* {vCCCC .. vNNNN}, meth@BBBB *op* {vCCCC .. vNNNN}, type@BBBB *(where NNNN = CCCC+AA-1, that is A determines the count 0..255, and C determines the first register)*	
AA|*op* BBBB CCCC	3rms	*op* {vCCCC .. vNNNN}, vtaboff@BBBB *(where NNNN = CCCC+AA-1, that is A determines the count 0..255, and C determines the first register)*	*(suggested format for statically linked invoke-virtual and invoke-super instructions of format 3rc)*
AA|*op* CCBB DDDD	3rfs	*op* {vDDDD .. vNNNN}, vtaboff@BB, iface@CC *(where NNNN = DDDD+AA-1, that is A determines the count 0..255, and D determines the first register)*	*(suggested format for statically linked invoke-interface instructions of format 3rc)*
AA|*op* $BBBB_{lo}$ BBBB BBBB $BBBB_{hi}$	51l	*op* vAA, #+BBBBBBBBBBBBBBBB	const-wide