

Dalvik 虚拟机操作码

作者: [Gabor Paller](#) 翻译: [YULIANGMAX](#)

v1.0

表中的 `vx`、`vy`、`vz` 表示某个 Dalvik 寄存器。根据不同指令可以访问 16、256 或 64K 寄存器。

表中 `lit4`、`lit8`、`lit16`、`lit32`、`lit64` 表示字面值（直接赋值），数字是值所占位数的长度。

`long` 和 `double` 型的值占用两个寄存器，例：一个在 `v0` 寄存器的 `double` 值实际占用 `v0,v1` 两个寄存器。

`boolean` 值的存储实际是 1 和 0，1 为真、0 为假；`boolean` 型的值实际是转成 `int` 型的值进行操作。

所有例子的字节序都采用高位存储格式，例：0F00 0A00 的编译为 0F, 00, 0A, 00 存储。

有一些指令没有说明和例子，因为我没有在正常使用中看到过这些指令，它们的存在是从这里知道的: [Android opcode constant list](#)。

Opcode 操作码(hex)	Opcode name 操作码名称	Explanation 说明	Example 示例
00	nop	无操作	0000 - nop
01	move vx, vy	移动 <code>vy</code> 的内容到 <code>vx</code> 。两个寄存器都必须在最初的 256 寄存器范围以内。	0110 - move v0, v1 移动 v1 寄存器中的内容到 v0。
02	move/from16 vx, vy	移动 <code>vy</code> 的内容到 <code>vx</code> 。 <code>vy</code> 可能在 64K 寄存器范围以内，而 <code>vx</code> 则是在最初的 256 寄存器范围以内。	0200 1900 - move/from16 v0, v25 移动 v25 寄存器中的内容到 v0。
03	move/16	未知 ^{注4}	
04	move-wide	未知 ^{注4}	
05	move-wide/from16 vx, vy	移动一个 <code>long/double</code> 值，从 <code>vy</code> 到 <code>vx</code> 。 <code>vy</code> 可能在 64K 寄存器范围以内，而 <code>vx</code> 则是在最初的 256 寄存器范围以内。	0516 0000 - move-wide/from16 v22, v0 移动 v0,v1 寄存器中的内容到 v22,v23。
06	move-wide/16	未知 ^{注4}	
07	move-object vx, vy	移动对象引用，从 <code>vy</code> 到 <code>vx</code> 。	0781 - move-object v1, v8 移动 v8 寄存器中的对象引用到 v1。
08	move-object/from16 vx, vy	移动对象引用，从 <code>vy</code> 到 <code>vx</code> 。 <code>vy</code> 可以处理 64K 寄存器地址， <code>vx</code> 可以处理 256 寄存器地址。	0801 1500 - move-object/from16 v1, v21 移动 v21 寄存器中的对象引用到 v1。
09	move-object/16	未知 ^{注4}	
0A	move-result vx	移动上一次方法调用的返回值到 <code>vx</code> 。	0A00 - move-result v0 移动上一次方法调用的返回值到 v0。
0B	move-result-wide vx	移动上一次方法调用的 <code>long/double</code> 型返回值到 <code>vx,vx+1</code> 。	0B02 - move-result-wide v2 移动上一次方法调用的 <code>long/double</code> 型返回值到 v2,v3。
0C	move-result-object vx	移动上一次方法调用的对象引用返回值到 <code>vx</code> 。	0C00 - move-result-object v0 移动上一次方法调用的对象引用返回值到 v0。
0D	move-exception vx	当方法调用抛出异常时移动异常对象引用到 <code>vx</code> 。	0D19 - move-exception v25 当方法调用抛出异常时移动异常对象引用到 v25。
0E	return-void	返回空值。	0E00 - return-void 返回值为 <code>void</code> ，即无返回值，并非返回 <code>null</code> 。

0F	return vx	返回在 vx 寄存器的值。	0F00 - return v0 返回 v0 寄存器中的值。
10	return-wide vx	返回在 vx,vx+1 寄存器的 double/long 值。	1000 - return-wide v0 返回 v0,v1 寄存器中的 double/long 值。
11	return-object vx	返回在 vx 寄存器的对象引用。	1100 - return-object v0 返回 v0 寄存器中的对象引用。
12	const/4 vx, lit4	存入 4 位常量到 vx。	1221 - const/4 v1, #int 2 存入 int 型常量 2 到 v1。目的寄存器在第二个字节的低 4 位，常量 2 在更高的 4 位。
13	const/16 vx, lit16	存入 16 位常量到 vx。	1300 0A00 - const/16 v0, #int 10 存入 int 型常量 10 到 v0。
14	const vx, lit32	存入 int 型常量到 vx。	1400 4E61 BC00 - const v0, #12345678 // #00BC614E 存入常量 12345678 到 v0。
15	const/high16 v0, lit16	存入 16 位常量到最高位寄存器，用于初始化 float 值。	1500 2041 - const/high16 v0, #float 10.0 // #41200000 存入 float 常量 10.0 到 v0。该指令最高支持 16 位浮点数。
16	const-wide/16 vx, lit16	存入 int 常量到 vx,vx+1 寄存器，扩展 int 型常量为 long 常量。	1600 0A00 - const-wide/16 v0, #long 10 存入 long 常量 10 到 v0,v1 寄存器。
17	const-wide/32 vx, lit32	存入 32 位常量到 vx,vx+1 寄存器，扩展 int 型常量到 long 常量。	1702 4e61 bc00 - const-wide/32 v2, #long 12345678 // #00bc614e 存入 long 常量 12345678 到 v2,v3 寄存器。
18	const-wide vx, lit64	存入 64 位常量到 vx,vx+1 寄存器。	1802 874b 6b5d 54dc 2b00- const-wide v2, #long 12345678901234567 // #002bdc545d6b4b87 存入 long 常量 12345678901234567 到 v2,v3 寄存器。
19	const-wide/high16 vx, lit16	存入 16 位常量到最高 16 位的 vx,vx+1 寄存器，用于初始化 double 值。	1900 2440 - const-wide/high16 v0, #double 10.0 // #402400000 存入 double 常量 10.0 到 v0,v1。
1A	const-string vx, 字符串 ID	存入字符串常量引用到 vx，通过字符串 ID 或字符串。	1A08 0000 - const-string v8, "" // string@0000 存入 string@0000（字符串表#0 条目）的引用到 v8。
1B	const-string-jumbo	未知 ^{注 4}	
1C	const-class vx, 类型 ID	存入类对象常量到 vx，通过类型 ID 或类型（如 Object.class）。	1C00 0100 - const-class v0, Test3 // type@0001 存入 Test3.class（类型 ID 表#1 条目）的引用到 v0。
1D	monitor-enter vx	获得 vx 寄存器中的对象引用的监视器。	1D03 - monitor-enter v3 获得 v3 寄存器中的对象引用的监视器。
1E	monitor-exit	释放 vx 寄存器中的对象引用的监视器。	1E03 - monitor-exit v3 释放 v3 寄存器中的对象引用的监视器。

1F	check-cast vx, 类型 ID	检查 vx 寄存器中的对象引用是否可以转换成类型 ID 对应类型的实例。如不可转换, 抛出 ClassCastException 异常, 否则继续执行。	1F04 0100 - check-cast v4, Test3 // type@0001 检查 v4 寄存器中的对象引用是否可以转换成 Test3 (类型 ID 表#1 条目) 的实例。
20	instance-of vx, vy, 类型 ID	检查 vy 寄存器中的对象引用是否是类型 ID 对应类型的实例, 如果是, vx 存入非 0 值, 否则 vx 存入 0。	2040 0100 - instance-of v0, v4, Test3 // type@0001 检查 v4 寄存器中的对象引用是否是 Test3 (类型 ID 表#1 条目) 的实例。如果是, v0 存入非 0 值, 否则 v0 存入 0。
21	array-length vx, vy	计算 vy 寄存器中数组引用的元素长度并将长度存入 vx。	2111 - array-length v0, v1 计算 v1 寄存器中数组引用的元素长度并将长度存入 v0。
22	new-instance vx, 类型 ID	根据类型 ID 或类型新建一个对象实例, 并将新建的对象的引用存入 vx。	2200 1500 - new-instance v0, java.io.FileInputStream // type@0015 实例化 java.io.FileInputStream (类型 ID 表#15H 条目) 类型, 并将其对象引用存入 v0。
23	new-array vx, vy, 类型 ID	根据类型 ID 或类型新建一个数组, vy 存入数组的长度, vx 存入数组的引用。	2312 2500 - new-array v2, v1, char[] // type@0025 新建一个 char (类型 ID 表#25H 条目) 数组, v1 存入数组的长度, v2 存入数组的引用。
24	filled-new-array {参数}, 类型 ID	根据类型 ID 或类型新建一个数组并通过参数填充 ^{注5} 。新的数组引用可以得到一个 move-result-object 指令, 前提是执行过 filled-new-array 指令。	2420 530D 0000 - filled-new-array {v0, v0}, [I // type@0D53 新建一个 int (类型 ID 表#D53H 条目) 数组, 长度将为 2 并且 2 个元素将填充到 v0 寄存器。
25	filled-new-array-range {vx..vy}, 类型 ID	根据类型 ID 或类型新建一个数组并以寄存器范围为参数填充。新的数组引用可以得到一个 move-result-object 指令, 前提是执行过 filled-new-array 指令。	2503 0600 1300 - filled-new-array/range {v19..v21}, [B // type@0006 新建一个 byte (类型 ID 表#6 条目) 数组, 长度将为 3 并且 3 个元素将填充到 v19, v20, v21 寄存器 ^{注4} 。
26	fill-array-data vx, 偏移量	用 vx 的静态数据填充数组引用。静态数据的位址是当前指令位置加偏移量的和。	2606 2500 0000 - fill-array-data v6, 00e6 // +0025 用当前指令位置+25H 的静态数据填充 v6 寄存器的数组引用。偏移量是 32 位的数字, 静态数据的存储格式如下: 0003 // 表类型: 静态数组数据 0400 // 每个元素的字节数 (这个例子是 4 字节的 int 型) 0300 0000 // 元素个数 0100 0000 // 元素 #0: int 1 0200 0000 // 元素 #1: int 2 0300 0000 // 元素 #2: int 3
27	throw vx	抛出异常对象, 异常对象的引用在 vx 寄存器。	2700 - throw v0 抛出异常对象, 异常对象的引用在 v0 寄存器。
28	goto 目标	通过短偏移量 ^{注2} 无条件跳转到目标。	28F0 - goto 0005 // -0010 跳转到当前位置-16 (hex 10) 的位置, 0005

			是目标指令标签。
29	goto/16 目标	通过 16 位偏移量 ^{注2} 无条件跳转到目标。	2900 0FFE - goto/16 002f // -01f1 跳转到当前位置-1F1H 的位置，002f 是目标指令标签。
2A	goto/32 目标	通过 32 位偏移量 ^{注2} 无条件跳转到目标。	
2B	packed-switch vx, 索引表偏移量	实现一个 switch 语句，case 常量是连续的。这个指令使用索引表，vx 是在表中找到具体 case 的指令偏移量的索引，如果无法在表中找到 vx 对应的索引将继续执行下一个指令（即 default case）。	2B02 0C00 0000 - packed-switch v2, 000c // +000c 根据 v2 寄存器中的值执行 packed switch，索引表的位置是当前指令位置+0CH，表如下所示： 0001 // 表类型：packed switch 表 0300 // 元素个数 0000 0000 // 基础元素 0500 0000 0: 00000005 // case 0: +0000 0005 0700 0000 1: 00000007 // case 1: +0000 0007 0900 0000 2: 00000009 // case 2: +0000 0009
2C	sparse-switch vx, 查询表偏移量	实现一个 switch 语句，case 常量是非连续的。这个指令使用查询表，用于表示 case 常量和每个 case 常量的偏移量。如果 vx 无法在表中匹配将继续执行下一个指令（即 default case）。	2C02 0c00 0000 - sparse-switch v2, 000c // +000c 根据 v2 寄存器中的值执行 sparse switch，查询表的位置是当前指令位置+0CH，表如下所示： 0002 // 表类型：sparse switch 表 0300 // 元素个数 9cff ffff // 第一个 case 常量：-100 fa00 0000 // 第二个 case 常量：250 e803 0000 // 第三个 case 常量：1000 0500 0000 // 第一个 case 常量的偏移量：+5 0700 0000 // 第二个 case 常量的偏移量：+7 0900 0000 // 第三个 case 常量的偏移量：+9
2D	cmpl-float vx, vy, vz	比较 vy 和 vz 的 float 值并在 vx 存入 int 型返回值 ^{注3} 。	2D00 0607 - cmpl-float v0, v6, v7 比较 v6 和 v7 的 float 值并在 v0 存入 int 型返回值。非数值默认为小于。如果参数为非数值将返回-1。
2E	cmpg-float vx, vy, vz	比较 vy 和 vz 的 float 值并在 vx 存入 int 型返回值 ^{注3} 。	2E00 0607 - cmpg-float v0, v6, v7 比较 v6 和 v7 的 float 值并在 v0 存入 int 型返回值。非数值默认为大于。如果参数为非数值将返回 1。
2F	cmpl-double vx, vy, vz	比较 vy 和 vz ^{注2} 的 double 值并在 vx 存入 int 型返回值 ^{注3} 。	2F19 0608 - cmpl-double v25, v6, v8 比较 v6,v7 和 v8,v9 的 double 值并在 v25 存入 int 型返回值。非数值默认为小于。如果参数为非数值将返回-1。
30	cmpg-double vx, vy, vz	比较 vy 和 vz ^{注2} 的 double 值并在 vx 存入 int 型返回值 ^{注3} 。	3000 080A - cmpg-double v0, v8, v10 比较 v8,v9 和 v10,v11 的 double 值并在 v0 存入 int 型返回值。非数值默认为大于。如果参数

			为非数值将返回 1。
31	cmp-long vx, vy, vz	比较 vx 和 vz 的 long 值并在 vx 存入 int 型返回值 ^{注3} 。	3100 0204 - cmp-long v0, v2, v4 比较 v2 和 v4 的 long 值并在 v0 存入 int 型返回值。
32	if-eq vx,vy, 目标	如果 vx == vy ^{注2} , 跳转到目标。 vx 和 vy 是 int 型值。	32b3 6600 - if-eq v3, v11, 0080 // +0066 如果 v3 == v11, 跳转到当前位置+66H。0080 是目标指令标签。
33	if-ne vx,vy, 目标	如果 vx != vy ^{注2} , 跳转到目标。 vx 和 vy 是 int 型值。	33A3 1000 - if-ne v3, v10, 002c // +0010 如果 v3 != v10, 跳转到当前位置+10H。002c 是目标指令标签。
34	if-lt vx,vy, 目标	如果 vx < vy ^{注2} , 跳转到目标。vx 和 vy 是 int 型值。	3432 CBFF - if-lt v2, v3, 0023 // -0035 如果 v2 < v3, 跳转到当前位置-35H。0023 是目标指令标签。
35	if-ge vx, vy, 目标	如果 vx >= vy ^{注2} , 跳转到目标。 vx 和 vy 是 int 型值。	3510 1B00 - if-ge v0, v1, 002b // +001b 如果 v0 >= v1, 跳转到当前位置+1BH。002b 是目标指令标签。
36	if-gt vx,vy, 目标	如果 vx > vy ^{注2} , 跳转到目标。vx 和 vy 是 int 型值。	3610 1B00 - if-ge v0, v1, 002b // +001b 如果 v0 > v1, 跳转到当前位置+1BH。002b 是目标指令标签。
37	if-le vx,vy, 目标	如果 vx <= vy ^{注2} , 跳转到目标。 vx 和 vy 是 int 型值。	3756 0B00 - if-le v6, v5, 0144 // +000b 如果 v6 <= v5, 跳转到当前位置+0BH。0144 是目标指令标签。
38	if-eqz vx, 目标	如果 vx == 0 ^{注2} , 跳转到目标。vx 是 int 型值。	3802 1900 - if-eqz v2, 0038 // +0019 如果 v2 == 0, 跳转到当前位置+19H。0038 是目标指令标签。
39	if-nez vx, 目标	如果 vx != 0 ^{注2} , 跳转到目标。	3902 1200 - if-nez v2, 0014 // +0012 如果 v2 != 0, 跳转到当前位置+18(hex 12)。0014 是目标指令标签。
3A	if-ltz vx, 目标	如果 vx < 0 ^{注2} , 跳转到目标。	3A00 1600 - if-ltz v0, 002d // +0016 如果 v0 < 0, 跳转到当前位置+16H。002d 是目标指令标签。
3B	if-gez vx, 目标	如果 vx >= 0 ^{注2} , 跳转到目标。	3B00 1600 - if-gez v0, 002d // +0016 如果 v0 >= 0, 跳转到当前位置+16H。002d 是目标指令标签。
3C	if-gtz vx, 目标	如果 vx > 0 ^{注2} , 跳转到目标。	3C00 1D00 - if-gtz v0, 004a // +001d 如果 v0 > 0, 跳转到当前位置+1DH。004a 是目标指令标签。
3D	if-lez vx, 目标	如果 vx <= 0 ^{注2} , 跳转到目标。	3D00 1D00 - if-lez v0, 004a // +001d 如果 v0 <= 0, 跳转到当前位置+1DH。004a 是目标指令标签。

3E	unused_3E	未使用	
3F	unused_3F	未使用	
40	unused_40	未使用	
41	unused_41	未使用	
42	unused_42	未使用	
43	unused_43	未使用	
44	aget vx, vy, vz	从 int 数组获取一个 int 型值到 vx, 对象数组的引用位于 vy, 需获取的元素的索引位于 vz。	4407 0306 - aget v7, v3, v6 从数组获取一个 int 型值到 v7, 对象数组的引用位于 v3, 需获取的元素的索引位于 v6。
45	aget-wide vx, vy, vz	从 long/double 数组获取一个 long/double 值到 vx, vx+1, 数组的引用位于 vy, 需获取的元素的索引位于 vz。	4505 0104 - aget-wide v5, v1, v4 从 long/double 数组获取一个 long/double 值到 v5, vx6, 数组的引用位于 v1, 需获取的元素的索引位于 v4。
46	aget-object vx, vy, vz	从对象引用数组获取一个对象引用到 vx, 对象数组的引用位于 vy, 需获取的元素的索引位于 vz。	4602 0200 - aget-object v2, v2, v0 从对象引用数组获取一个对象引用到 v2, 对象数组的引用位于 v2, 需获取的元素的索引位于 v0。
47	aget-boolean vx, vy, vz	从 boolean 数组获取一个 boolean 值到 vx, 数组的引用位于 vy, 需获取的元素的索引位于 vz。	4700 0001 - aget-boolean v0, v0, v1 从 boolean 数组获取一个 boolean 值到 v0, 数组的引用位于 v0, 需获取的元素的索引位于 v1。
48	aget-byte vx, vy, vz	从 byte 数组获取一个 byte 值到 vx, 数组的引用位于 vy, 需获取的元素的索引位于 vz。	4800 0001 - aget-byte v0, v0, v1 从 byte 数组获取一个 byte 值到 v0, 数组的引用位于 v0, 需获取的元素的索引位于 v1。
49	aget-char vx, vy, vz	从 char 数组获取一个 char 值到 vx, 数组的引用位于 vy, 需获取的元素的索引位于 vz。	4905 0003 - aget-char v5, v0, v3 从 char 数组获取一个 char 值到 v5, 数组的引用位于 v0, 需获取的元素的索引位于 v3。
4A	aget-short vx, vy, vz	从 short 数组获取一个 short 值到 vx, 数组的引用位于 vy, 需获取的元素的索引位于 vz。	4A00 0001 - aget-short v0, v0, v1 从 short 数组获取一个 short 值到 v0, 数组的引用位于 v0, 需获取的元素的索引位于 v1。
4B	aput vx, vy, vz	将 vx 的 int 值作为元素存入 int 数组, 数组的引用位于 vy, 元素的索引位于 vz。	4B00 0305 - aput v0, v3, v5 将 v0 的 int 值作为元素存入 int 数组, 数组的引用位于 v3, 元素的索引位于 v5。
4C	aput-wide vx, vy, vz	将 vx, vx+1 的 double/long 值作为元素存入 double/long 数组, 数组的引用位于 vy, 元素的索引位于 vz。	4C05 0104 - aput-wide v5, v1, v4 将 v5, v6 的 double/long 值作为元素存入 double/long 数组, 数组的引用位于 v1, 元素的索引位于 v4。
4D	aput-object vx, vy, vz	将 vx 的对象引用作为元素存入对象引用数组, 数组的引用位于 vy, 元素的索引位于 vz。	4D02 0100 - aput-object v2, v1, v0 将 v2 的对象引用作为元素存入对象引用数组, 数组的引用位于 v1, 元素的索引位于 v0。
4E	aput-boolean vx, vy, vz	将 vx 的 boolean 值作为元素存入 boolean 数组, 数组的引用位于 vy, 元素的索引位于 vz。	4E01 0002 - aput-boolean v1, v0, v2 将 v1 的 boolean 值作为元素存入 boolean 数组, 数组的引用位于 v0, 元素的索引位于 v2。
4F	aput-byte vx, vy, vz	将 vx 的 byte 值作为元素存入 byte 数组, 数组的引用位于 vy, 元素的索引位于 vz。	4F02 0001 - aput-byte v2, v0, v1 将 v2 的 byte 值作为元素存入 byte 数组, 数组的引用位于 v0, 元素的索引位于 v1。

		素的索引位于 vz 。	的引用位于 v0 ，元素的索引位于 v1 。
50	aput-char vx, vy, vz	将 vx 的 char 值作为元素存入 char 数组，数组的引用位于 vy，元素的索引位于 vz。	5003 0001 - aput-char v3, v0, v1 将 v3 的 char 值作为元素存入 char 数组，数组的引用位于 v0，元素的索引位于 v1。
51	aput-short vx, vy, vz	将 vx 的 short 值作为元素存入 short 数组，数组的引用位于 vy，元素的索引位于 vz。	5102 0001 - aput-short v2, v0, v1 将 v2 的 short 值作为元素存入 short 数组，数组的引用位于 v0，元素的索引位于 v1。
52	iget vx, vy, 字段 ID	根据 字段 ID 读取实例的 int 型字段到 vx, vy 寄存器中是该实例的引用。	5210 0300 - iget v0, v1, Test2.i6:I // field@0003 读取 int 型字段 i6（字段表#3 条目）到 v0, v1 寄存器中是 Test2 实例的引用。
53	iget-wide vx, vy, 字段 ID	根据 字段 ID 读取实例的 double/long 型字段到 vx, vx+1 ^{注 1} , vy 寄存器中是该实例的引用。	5320 0400 - iget-wide v0, v2, Test2.l0:J // field@0004 读取 long 型字段 l0（字段表#4 条目）到 v0, v1, v2 寄存器中是 Test2 实例的引用。
54	iget-object vx, vy, 字段 ID	根据 字段 ID 读取一个实例的对象引用字段到 vx, vy 寄存器中是该实例的引用。	iget-object v1, v2, LineReader.fis:Ljava/io/FileInputStream; // field@0002 读取 FileInputStream 对象引用字段 fis（字段表#2 条目）到 v1, v2 寄存器中是 LineReader 实例的引用。
55	iget-boolean vx, vy, 字段 ID	根据 字段 ID 读取实例的 boolean 型字段到 vx, vy 寄存器中是该实例的引用。	55FC 0000 - iget-boolean v12, v15, Test2.b0:Z // field@0000 读取 boolean 型字段 b0（字段表#0 条目）到 v12, v15 寄存器中是 Test2 实例的引用。
56	iget-byte vx, vy, 字段 ID	根据 字段 ID 读取实例的 byte 型字段到 vx, vy 寄存器中是该实例的引用。	5632 0100 - iget-byte v2, v3, Test3.bi1:B // field@0001 读取 byte 型字段 bi1（字段表#1 条目）到 v2, v3 寄存器中是 Test2 实例的引用。
57	iget-char vx, vy, 字段 ID	根据 字段 ID 读取实例的 char 型字段到 vx, vy 寄存器中是该实例的引用。	5720 0300 - iget-char v0, v2, Test3.ci1:C // field@0003 读取 char 型字段 ci1（字段表#3 条目）到 v0, v2 寄存器中是 Test2 实例的引用。
58	iget-short vx, vy, 字段 ID	根据 字段 ID 读取实例的 short 型字段到 vx, vy 寄存器中是该实例的引用。	5830 0800 - iget-short v0, v3, Test3.si1:S // field@0008 读取 short 型字段 si1（字段表#8 条目）到 v0, v3 寄存器中是 Test2 实例的引用。
59	iput vx, vy, 字段 ID	根据 字段 ID 将 vx 寄存器的值存入实例的 int 型字段，vy 寄存器中是该实例的引用。	5920 0200 - iput v0, v2, Test2.i6:I // field@0002 将 v0 寄存器的值存入实例的 int 型字段 i6（字段表#2 条目），v2 寄存器中是 Test2 实例的引用。
5A	iput-wide vx, vy, 字段 ID	根据 字段 ID 将 vx, vx+1 寄存器的值存入实例的 double/long 型字段，vy 寄存器中是该实例的引用。	5A20 0000 - iput-wide v0, v2, Test2.d0:D // field@0000 将 v0, v1 寄存器的值存入实例的 double 型字段 d0（字段表#0 条目），v2 寄存器中是 Test2 实例的引用。

5B	iput-object vx, vy, 字段ID	根据 字段 ID 将 vx 寄存器的值存入实例的对象引用字段, vy 寄存器中是该实例的引用。	5B20 0000 - iput-object v0, v2, LineReader.bis:Ljava/io/BufferedInputStream; // field@0000 将 v0 寄存器的值存入实例的对象引用字段 bis (字段表#0 条目), v2 寄存器中是 BufferedInputStream 实例的引用。
5C	iput-boolean vx, vy, 字段ID	根据 字段 ID 将 vx 寄存器的值存入实例的 boolean 型字段, vy 寄存器中是该实例的引用。	5C30 0000 - iput-boolean v0, v3, Test2.b0:Z // field@0000 将 v0 寄存器的值存入实例的 boolean 型字段 b0 (字段表#0 条目), v3 寄存器中是 Test2 实例的引用。
5D	iput-byte vx, vy, 字段ID	根据 字段 ID 将 vx 寄存器的值存入实例的 byte 型字段, vy 寄存器中是该实例的引用。	5D20 0100 - iput-byte v0, v2, Test3.bi1:B // field@0001 将 v0 寄存器的值存入实例的 byte 型字段 bi1 (字段表#1 条目), v2 寄存器中是 Test2 实例的引用。
5E	iput-char vx, vy, 字段ID	根据 字段 ID 将 vx 寄存器的值存入实例的 char 型字段, vy 寄存器中是该实例的引用。	5E20 0300 - iput-char v0, v2, Test3.ci1:C // field@0003 将 v0 寄存器的值存入实例的 char 型字段 ci1 (字段表#3 条目), v2 寄存器中是 Test2 实例的引用。
5F	iput-short vx, vy, 字段ID	根据 字段 ID 将 vx 寄存器的值存入实例的 short 型字段, vy 寄存器中是该实例的引用。	5F21 0800 - iput-short v1, v2, Test3.si1:S // field@0008 将 v0 寄存器的值存入实例的 short 型字段 si1 (字段表#8 条目), v2 寄存器中是 Test2 实例的引用。
60	sget vx, 字段ID	根据 字段 ID 读取静态 int 型字段到 vx。	6000 0700 - sget v0, Test3.is1:I // field@0007 读取 Test3 的静态 int 型字段 is1 (字段表#7 条目) 到 v0。
61	sget-wide vx, 字段ID	根据 字段 ID 读取静态 double/long 型字段到 vx, vx+1。	6100 0500 - sget-wide v0, Test2.l1:J // field@0005 读取 Test2 的静态 long 型字段 l1 (字段表#5 条目) 到 v0, v1。
62	sget-object vx, 字段ID	根据 字段 ID 读取静态对象引用字段到 vx。	6201 0C00 - sget-object v1, Test3.os1:Ljava/lang/Object; // field@000c 读取 Object 的静态对象引用字段 os1 (字段表#CH 条目) 到 v1。
63	sget-boolean vx, 字段ID	根据 字段 ID 读取静态 boolean 型字段到 vx。	6300 0C00 - sget-boolean v0, Test2.sb:Z // field@000c 读取 Test2 的静态 boolean 型字段 sb (字段表#CH 条目) 到 v0。
64	sget-byte vx, 字段ID	根据 字段 ID 读取静态 byte 型字段到 vx。	6400 0200 - sget-byte v0, Test3.bs1:B // field@0002 读取 Test3 的静态 byte 型字段 bs1 (字段表#2 条目) 到 v0。
65	sget-char vx, 字段ID	根据 字段 ID 读取静态 char 型字段到 vx。	6500 0700 - sget-char v0, Test3.cs1:C

	字段 ID	段到 vx。	// field@0007 读取 Test3 的静态 char 型字段 cs1（字段表#7 条目）到 v0。
66	sget-short vx, 字段 ID	根据 字段 ID 读取静态 short 型字段到 vx。	6600 0B00 - sget-short v0, Test3.ss1:S // field@000b 读取 Test3 的静态 short 型字段 ss1（字段表#CH 条目）到 v0。
67	sput vx, 字段 ID	根据 字段 ID 将 vx 寄存器中的值赋值到 int 型静态字段。	6700 0100 - sput v0, Test2.i5:I // field@0001 将 v0 寄存器中的值赋值到 Test2 的 int 型静态字段 i5（字段表#1 条目）。
68	sput-wide vx, 字段 ID	根据 字段 ID 将 vx, vx+1 寄存器中的值赋值到 double/long 型静态字段。	6800 0500 - sput-wide v0, Test2.l1:J // field@0005 将 v0, v1 寄存器中的值赋值到 Test2 的 long 型静态字段 l1（字段表#5 条目）。
69	sput-object vx, 字段 ID	根据 字段 ID 将 vx 寄存器中的对象引用赋值到对象引用静态字段。	6900 0c00 - sput-object v0, Test3.os1:Ljava/lang/Object; // field@000c 将 v0 寄存器中的对象引用赋值到 Test3 的对象引用静态字段 os1（字段表#CH 条目）。
6A	sput-boolean vx, 字段 ID	根据 字段 ID 将 vx 寄存器中的值赋值到 boolean 型静态字段。	6A00 0300 - sput-boolean v0, Test3.bls1:Z // field@0003 将 v0 寄存器中的值赋值到 Test3 的 boolean 型静态字段 bls1（字段表#3 条目）。
6B	sput-byte vx, 字段 ID	根据 字段 ID 将 vx 寄存器中的值赋值到 byte 型静态字段。	6B00 0200 - sput-byte v0, Test3.bs1:B // field@0002 将 v0 寄存器中的值赋值到 Test3 的 byte 型静态字段 bs1（字段表#2 条目）。
6C	sput-char vx, 字段 ID	根据 字段 ID 将 vx 寄存器中的值赋值到 char 型静态字段。	6C01 0700 - sput-char v1, Test3.cs1:C // field@0007 将 v1 寄存器中的值赋值到 Test3 的 char 型静态字段 cs1（字段表#7 条目）。
6D	sput-short vx, 字段 ID	根据 字段 ID 将 vx 寄存器中的值赋值到 short 型静态字段。	6D00 0B00 - sput-short v0, Test3.ss1:S // field@000b 将 v0 寄存器中的值赋值到 Test3 的 short 型静态字段 ss1（字段表#BH 条目）。
6E	invoke-virtual {参数}, 方法名	调用带参数的虚拟方法。	6E53 0600 0421 - invoke-virtual { v4, v0, v1, v2, v3}, Test2.method5:(IIII)V // method@0006 调用 Test2 的 method5（方法表#6 条目）方法，该指令共有 5 个参数（操作码第二个字节的 4 个最高有效位 5） ^{注 5} 。参数 v4 是 "this" 实例，v0, v1, v2, v3 是 method5 方法的参数，(IIII)V 的 4 个 I 分表表示 4 个 int 型参数，V 表示返回值为 void。
6F	invoke-super {参数}, 方法名	调用带参数的直接父类的虚拟方法。	6F10 A601 0100 invoke-super {v1}, java.io.FilterOutputStream.close:()V // method@01a6

			调用 java.io.FilterOutputStream 的 close（方法表#1A6 条目）方法，参数 v1 是"this"实例。()V 表示 close 方法没有参数，V 表示返回值为 void。
70	invoke-direct {参数}, 方法名	不解析直接调用带参数的方法。	7010 0800 0100 - invoke-direct {v1}, java.lang.Object.<init>:()V // method@0008 调用 java.lang.Object 的<init>（方法表#8 条目）方法，参数 v1 是"this"实例 ^{注5} 。()V 表示<init>方法没有参数，V 表示返回值为 void。
71	invoke-static {参数}, 方法名	调用带参数的静态方法。	7110 3400 0400 - invoke-static {v4}, java.lang.Integer.parseInt:(Ljava/lang/String;)I // method@0034 调用 java.lang.Integer 的 parseInt（方法表#34 条目）静态方法，该指令只有 1 个参数 v4 ^{注5} ，(Ljava/lang/String;)I 中的 Ljava/lang/String; 表示 parseInt 方法需要 String 类型的参数，I 表示返回值为 int 型。
72	invoke-interface {参数}, 方法名	调用带参数的接口方法。	7240 2102 3154 invoke-interface {v1, v3, v4, v5}, mfwf.IReceivingProtocolAdapter.receivePackage:(ILjava/lang/String;Ljava/io/InputStream;)Z // method@0221 调用 mfwf.IReceivingProtocolAdapter 接口的 receivePackage 方法（方法表#221 条目），该指令共有 4 个参数 ^{注5} ，参数 v1 是"this"实例，v3,v4,v5 是 receivePackage 方法的参数，(ILjava/lang/String;Ljava/io/InputStream;)Z 中的 I 表示 int 型参数，Ljava/lang/String; 表示 String 类型参数，Ljava/io/InputStream; 表示 InputStream 类型参数，Z 表示返回值为 boolean 型。
73	unused_73	未使用	
74	invoke-virtual/range {vx..vy}, 方法名	调用以寄存器范围为参数的虚拟方法。该指令第一个寄存器和寄存器的数量将传递给方法。	7403 0600 1300 - invoke-virtual {v19..v21}, Test2.method5:(IIII)V // method@0006 调用 Test2 的 method5（方法表#6 条目）方法，该指令共有 3 个参数。参数 v19 是"this"实例，v20,v21 是 method5 方法的参数，(IIII)V 的 4 个 I 分表表示 4 个 int 型参数，V 表示返回值为 void。
75	invoke-super/range {vx..vy}, 方法名	调用以寄存器范围为参数的直接父类的虚拟方法。该指令第一个寄存器和寄存器的数量将会传递给方法。	7501 A601 0100 invoke-super {v1}, java.io.FilterOutputStream.close:()V // method@01a6 调用 java.io.FilterOutputStream 的 close（方法表#1A6 条目）方法，参数 v1 是"this"实例。()V 表示 close 方法没有参数，V 表示返

			回值为 void。
76	invoke-direct/ range {vx..vy}, 方法名	不解析直接调用以寄存器范围为参数的方法。该指令第一个寄存器和寄存器的数量将会传递给方法。	7603 3A00 1300 - invoke-direct/range {v19..21}, java.lang.Object.<init>:()V // method@003a 调用 java.lang.Object 的<init>（方法表#3A 条目）方法，参数 v19 是"this"实例（操作码第五、第六字节表示范围从 v19 开始，第二个字节为 03 表示传入了 3 个参数），()V 表示<init>方法没有参数，V 表示返回值为 void。
77	invoke-static/ range {vx..vy}, 方法名	调用以寄存器范围为参数的静态方法。该指令第一个寄存器和寄存器的数量将会传递给方法。	7703 3A00 1300 - invoke-static/range {v19..21}, java.lang.Integer.parseInt:(Ljava/lang/String;)I // method@0034 调用 java.lang.Integer 的 parseInt（方法表#34 条目）静态方法，参数 v19 是"this"实例（操作码第五、第六字节表示范围从 v19 开始，第二个字节为 03 表示传入了 3 个参数），(Ljava/lang/String;)I 中的 Ljava/lang/String; 表示 parseInt 方法需要 String 类型的参数，I 表示返回值为 int 型。
78	invoke-interface-range {vx..vy}, 方法名	调用以寄存器范围为参数的接口方法。该指令第一个寄存器和寄存器的数量将会传递给方法。	7840 2102 0100 invoke-interface {v1..v4}, mfwf.IReceivingProtocolAdapter.receivePackage:(ILjava/lang/String;Ljava/io/InputStream;)Z // method@0221 调用 mfwf.IReceivingProtocolAdapter 接口的 receivePackage 方法（方法表#221 条目），该指令共有 4 个参数 ^{注5} ，参数 v1 是"this"实例，v2, v3, v4 是 receivePackage 方法的参数，(ILjava/lang/String;Ljava/io/InputStream;)Z 中的 I 表示 int 型参数，Ljava/lang/String; 表示 String 类型参数，Ljava/io/InputStream; 表示 InputStream 类型参数，Z 表示返回值为 boolean 型。
79	unused_79	未使用	
7A	unused_7A	未使用	
7B	neg-int vx, vy	计算 vx = -vy 并将结果存入 vx。	7B01 - neg-int v1, v0 计算 -v0 并将结果存入 v1。
7C	not-int vx, vy	未知 ^{注4}	
7D	neg-long vx, vy	计算 vx, vx+1 = -(vy, vy+1) 并将结果存入 vx, vx+1。	7D02 - neg-long v2, v0 计算 -(v0, v1) 并将结果存入 (v2, v3)。
7E	not-long vx, vy	未知 ^{注4}	
7F	neg-float vx, vy	计算 vx = -vy 并将结果存入 vx。	7F01 - neg-float v1, v0 计算 -v0 并将结果存入 v1。
80	neg-double vx, vy	计算 vx, vx+1 = -(vy, vy+1) 并将结果存入 vx, vx+1。	8002 - neg-double v2, v0 计算 -(v0, v1) 并将结果存入 (v2, v3)。
81	int-to-long v	转换 vy 寄存器中的 int 型值为 long	8106 - int-to-long v6, v0

	x, vy	ong 型值存入 vx,vx+1。	转换 v0 寄存器中的 int 型值为 long 型值存入 v6,v7。
82	int-to-float vx, vy	转换 vy 寄存器中的 int 型值为 float 型值存入 vx。	8206 - int-to-float v6, v0 转换 v0 寄存器中的 int 型值为 float 型值存入 v6。
83	int-to-double vx, vy	转换 vy 寄存器中的 int 型值为 double 型值存入 vx,vx+1。	8306 - int-to-double v6, v0 转换 v0 寄存器中的 int 型值为 double 型值存入 v6,v7。
84	long-to-int vx, vy	转换 vy,vy+1 寄存器中的 long 型值为 int 型值存入 vx。	8424 - long-to-int v4, v2 转换 v2,v3 寄存器中的 long 型值为 int 型值存入 v4。
85	long-to-float vx, vy	转换 vy,vy+1 寄存器中的 long 型值为 float 型值存入 vx。	8510 - long-to-float v0, v1 转换 v1,v2 寄存器中的 long 型值为 float 型值存入 v0。
86	long-to-double vx, vy	转换 vy,vy+1 寄存器中的 long 型值为 double 型值存入 vx,vx+1。	8610 - long-to-double v0, v1 转换 v1,v2 寄存器中的 long 型值为 double 型值存入 v0,v1。
87	float-to-int vx, vy	转换 vy 寄存器中的 float 型值为 int 型值存入 vx。	8730 - float-to-int v0, v3 转换 v3 寄存器中的 float 型值为 int 型值存入 v0。
88	float-to-long vx, vy	转换 vy 寄存器中的 float 型值为 long 型值存入 vx,vx+1。	8830 - float-to-long v0, v3 转换 v3 寄存器中的 float 型值为 long 型值存入 v0,v1。
89	float-to-double vx, vy	转换 vy 寄存器中的 float 型值为 double 型值存入 vx,vx+1。	8930 - float-to-double v0, v3 转换 v3 寄存器中的 float 型值为 double 型值存入 v0,v1。
8A	double-to-int vx, vy	转换 vy,vy+1 寄存器中的 double 型值为 int 型值存入 vx。	8A40 - double-to-int v0, v4 转换 v4,v5 寄存器中的 double 型值为 int 型值存入 v0。
8B	double-to-long vx, vy	转换 vy,vy+1 寄存器中的 double 型值为 long 型值存入 vx,vx+1。	8B40 - double-to-long v0, v4 转换 v4,v5 寄存器中的 double 型值为 long 型值存入 v0,v1。
8C	double-to-float vx, vy	转换 vy,vy+1 寄存器中的 double 型值为 float 型值存入 vx。	8C40 - double-to-float v0, v4 转换 v4,v5 寄存器中的 double 型值为 float 型值存入 v0。
8D	int-to-byte vx, vy	转换 vy 寄存器中的 int 型值为 byte 型值存入 vx。	8D00 - int-to-byte v0, v0 转换 v0 寄存器中的 int 型值为 byte 型值存入 v0。
8E	int-to-char vx, vy	转换 vy 寄存器中的 int 型值为 char 型值存入 vx。	8E33 - int-to-char v3, v3 转换 v3 寄存器中的 int 型值为 char 型值存入 v3。
8F	int-to-short vx, vy	转换 vy 寄存器中的 int 型值为 short 型值存入 vx。	8F00 - int-to-short v3, v0 转换 v0 寄存器中的 int 型值为 short 型值存入 v0。
90	add-int vx, vy, vz	计算 vy + vz 并将结果存入 vx。	9000 0203 - add-int v0, v2, v3 计算 v2 + v3 并将结果存入 v0 ^{注4} 。

91	sub-int vx, vy, vz	计算 $vy - vz$ 并将结果存入 vx。	9100 0203 - sub-int v0, v2, v3 计算 $v2 - v3$ 并将结果存入 v0。
92	mul-int vx, vy, vz	计算 $vy * vz$ 并将结果存入 vx。	9200 0203 - mul-int v0, v2, v3 计算 $v2 * v3$ 并将结果存入 v0。
93	div-int vx, vy, vz	计算 vy / vz 并将结果存入 vx。	9303 0001 - div-int v3, v0, v1 计算 $v0 / v1$ 并将结果存入 v3。
94	rem-int vx, vy, vz	计算 $vy \% vz$ 并将结果存入 vx。	9400 0203 - rem-int v0, v2, v3 计算 $v3 \% v2$ 并将结果存入 v0。
95	and-int vx, vy, vz	计算 vy 与 vz 并将结果存入 vx。	9503 0001 - and-int v3, v0, v1 计算 v0 与 v1 并将结果存入 v3。
96	or-int vx, vy, vz	计算 vy 或 vz 并将结果存入 vx。	9603 0001 - or-int v3, v0, v1 计算 v0 或 v1 并将结果存入 v3。
97	xor-int vx, vy, vz	计算 vy 异或 vz 并将结果存入 vx。	9703 0001 - xor-int v3, v0, v1 计算 v0 异或 v1 并将结果存入 v3。
98	shl-int vx, vy, vz	左移 vy, vz 指定移动的位置, 结果存入 vx。	9802 0001 - shl-int v2, v0, v1 以 v1 指定的位置左移 v0, 结果存入 v2。
99	shr-int vx, vy, vz	右移 vy, vz 指定移动的位置, 结果存入 vx。	9902 0001 - shr-int v2, v0, v1 以 v1 指定的位置右移 v0, 结果存入 v2。
9A	ushr-int vx, vy, vz	无符号右移 vy, vz 指定移动的位置, 结果存入 vx。	9A02 0001 - ushr-int v2, v0, v1 以 v1 指定的位置无符号右移 v0, 结果存入 v2。
9B	add-long vx, vy, vz	计算 $vy, vy+1 + vz, vz+1$ 并将结果存入 vx, vx+1 ^{注1} 。	9B00 0305 - add-long v0, v3, v5 计算 $v3, v4 + v5, v6$ 并将结果存入 v0, v1。
9C	sub-long vx, vy, vz	计算 $vy, vy+1 - vz, vz+1$ 并将结果存入 vx, vx+1 ^{注1} 。	9C00 0305 - sub-long v0, v3, v5 计算 $v3, v4 - v5, v6$ 并将结果存入 v0, v1。
9D	mul-long vx, vy, vz	计算 $vy, vy+1 * vz, vz+1$ 并将结果存入 vx, vx+1 ^{注1} 。	9D00 0305 - mul-long v0, v3, v5 计算 $v3, v4 * v5, v6$ 并将结果存入 v0, v1。
9E	div-long vx, vy, vz	计算 $vy, vy+1 / vz, vz+1$ 并将结果存入 vx, vx+1 ^{注1} 。	9E06 0002 - div-long v6, v0, v2 计算 $v0, v1 / v2, v3$ 并将结果存入 v6, v7。
9F	rem-long vx, vy, vz	计算 $vy, vy+1 \% vz, vz+1$ 并将结果存入 vx, vx+1 ^{注1} 。	9F06 0002 - rem-long v6, v0, v2 计算 $v0, v1 \% v2, v3$ 并将结果存入 v6, v7。
A0	and-long vx, vy, vz	计算 $vy, vy+1$ 与 $vz, vz+1$ 并将结果存入 vx, vx+1 ^{注1} 。	A006 0002 - and-long v6, v0, v2 计算 v0, v1 与 v2, v3 并将结果存入 v6, v7。
A1	or-long vx, vy, vz	计算 $vy, vy+1$ 或 $vz, vz+1$ 并将结果存入 vx, vx+1 ^{注1} 。	A106 0002 - or-long v6, v0, v2 计算 v0, v1 或 v2, v3 并将结果存入 v6, v7。
A2	xor-long vx, vy, vz	计算 $vy, vy+1$ 异或 $vz, vz+1$ 并将结果存入 vx, vx+1 ^{注1} 。	A206 0002 - xor-long v6, v0, v2 计算 v0, v1 异或 v2, v3 并将结果存入 v6, v7。
A3	shl-long vx, vy, vz	左移 $vy, vy+1$, vz 指定移动的位置, 结果存入 vx, vx+1 ^{注1} 。	A302 0004 - shl-long v2, v0, v4 以 v4 指定的位置左移 v0, v1, 结果存入 v2, v3。
A4	shr-long vx, vy, vz	右移 $vy, vy+1$, vz 指定移动的位置, 结果存入 vx, vx+1 ^{注1} 。	A402 0004 - shr-long v2, v0, v4 以 v4 指定的位置右移 v0, v1, 结果存入 v2, v3。
A5	ushr-long vx, vy, vz	无符号右移 $vy, vy+1$, vz 指定移动的位置, 结果存入 vx, vx+1 ^{注1} 。	A502 0004 - ushr-long v2, v0, v4 以 v4 指定的位置无符号右移 v0, v1, 结果存入 v2, v3。
A6	add-float vx, vy, vz	计算 $vy + vz$ 并将结果存入 vx。	A600 0203 - add-float v0, v2, v3

	vy, vz		计算 $v2 + v3$ 并将结果存入 $v0$ 。
A7	sub-float vx, vy, vz	计算 $vy - vz$ 并将结果存入 vx 。	A700 0203 - sub-float $v0, v2, v3$ 计算 $v2 - v3$ 并将结果存入 $v0$ 。
A8	mul-float vx, vy, vz	计算 $vy * vz$ 并将结果存入 vx 。	A803 0001 - mul-float $v3, v0, v1$ 计算 $v0 * v1$ 并将结果存入 $v3$ 。
A9	div-float vx, vy, vz	计算 vy / vz 并将结果存入 vx 。	A903 0001 - div-float $v3, v0, v1$ 计算 $v0 / v1$ 并将结果存入 $v3$ 。
AA	rem-float vx, vy, vz	计算 $vy \% vz$ 并将结果存入 vx 。	AA03 0001 - rem-float $v3, v0, v1$ 计算 $v0 \% v1$ 并将结果存入 $v3$ 。
AB	add-double vx, vy, vz	计算 $vy, vy+1 + vz, vz+1$ 并将结果存入 $vx, vx+1$ ^{注1} 。	AB00 0305 - add-double $v0, v3, v5$ 计算 $v3, v4 + v5, v6$ 并将结果存入 $v0, v1$ 。
AC	sub-double vx, vy, vz	计算 $vy, vy+1 - vz, vz+1$ 并将结果存入 $vx, vx+1$ ^{注1} 。	AC00 0305 - sub-double $v0, v3, v5$ 计算 $v3, v4 - v5, v6$ 并将结果存入 $v0, v1$ 。
AD	mul-double vx, vy, vz	计算 $vy, vy+1 * vz, vz+1$ 并将结果存入 $vx, vx+1$ ^{注1} 。	AD06 0002 - mul-double $v6, v0, v2$ 计算 $v0, v1 * v2, v3$ 并将结果存入 $v6, v7$ 。
AE	div-double vx, vy, vz	计算 $vy, vy+1 / vz, vz+1$ 并将结果存入 $vx, vx+1$ ^{注1} 。	AE06 0002 - div-double $v6, v0, v2$ 计算 $v0, v1 / v2, v3$ 并将结果存入 $v6, v7$ 。
AF	rem-double vx, vy, vz	计算 $vy, vy+1 \% vz, vz+1$ 并将结果存入 $vx, vx+1$ ^{注1} 。	AF06 0002 - rem-double $v6, v0, v2$ 计算 $v0, v1 \% v2, v3$ 并将结果存入 $v6, v7$ 。
B0	add-int/2addr vx, vy	计算 $vx + vy$ 并将结果存入 vx 。	B010 - add-int/2addr $v0, v1$ 计算 $v0 + v1$ 并将结果存入 $v0$ 。
B1	sub-int/2addr vx, vy	计算 $vx - vy$ 并将结果存入 vx 。	B140 - sub-int/2addr $v0, v4$ 计算 $v0 - v4$ 并将结果存入 $v0$ 。
B2	mul-int/2addr vx, vy	计算 $vx * vy$ 并将结果存入 vx 。	B210 - mul-int/2addr $v0, v1$ 计算 $v0 * v1$ 并将结果存入 $v0$ 。
B3	div-int/2addr vx, vy	计算 vx / vy 并将结果存入 vx 。	B310 - div-int/2addr $v0, v1$ 计算 $v0 / v1$ 并将结果存入 $v0$ 。
B4	rem-int/2addr vx, vy	计算 $vx \% vy$ 并将结果存入 vx 。	B410 - rem-int/2addr $v0, v1$ 计算 $v0 \% v1$ 并将结果存入 $v0$ 。
B5	and-int/2addr vx, vy	计算 vx 与 vy 并将结果存入 vx 。	B510 - and-int/2addr $v0, v1$ 计算 $v0$ 与 $v1$ 并将结果存入 $v0$ 。
B6	or-int/2addr vx, vy	计算 vx 或 vy 并将结果存入 vx 。	B610 - or-int/2addr $v0, v1$ 计算 $v0$ 或 $v1$ 并将结果存入 $v0$ 。
B7	xor-int/2addr vx, vy	计算 vx 异或 vy 并将结果存入 vx 。	B710 - xor-int/2addr $v0, v1$ 计算 $v0$ 异或 $v1$ 并将结果存入 $v0$ 。
B8	shl-int/2addr vx, vy	左移 vx, vy 指定移动的位置, 并将结果存入 vx 。	B810 - shl-int/2addr $v0, v1$ 以 $v1$ 指定的位置左移 $v0$, 结果存入 $v0$ 。
B9	shr-int/2addr vx, vy	右移 vx, vy 指定移动的位置, 并将结果存入 vx 。	B910 - shr-int/2addr $v0, v1$ 以 $v1$ 指定的位置右移 $v0$, 结果存入 $v0$ 。
BA	ushr-int/2addr vx, vy	无符号右移 vx, vy 指定移动的位置, 并将结果存入 vx 。	BA10 - ushr-int/2addr $v0, v1$ 以 $v1$ 指定的位置无符号右移 $v0$, 结果存入 $v0$ 。
BB	add-long/2addr vx, vy	计算 $vx, vx+1 + vy, vy+1$ 并将结果存入 $vx, vx+1$ ^{注1} 。	BB20 - add-long/2addr $v0, v2$ 计算 $v0, v1 + v2, v3$ 并将结果存入 $v0, v1$ 。
BC	sub-long/2addr vx, vy	计算 $vx, vx+1 - vy, vy+1$ 并将结果存入 $vx, vx+1$ ^{注1} 。	BC70 - sub-long/2addr $v0, v7$

	vx, vy	果存入 vx, vx+1 ^{注1} 。	计算 v0, v1 - v7, v8 并将结果存入 v0, v1。
BD	mul-long/2addr vx, vy	计算 vx, vx+1 * vy, vy+1 并将结果存入 vx, vx+1 ^{注1} 。	BD70 - mul-long/2addr v0, v7 计算 v0, v1 * v7, v8 并将结果存入 v0, v1。
BE	div-long/2addr vx, vy	计算 vx, vx+1 / vy, vy+1 并将结果存入 vx, vx+1 ^{注1} 。	BE20 - div-long/2addr v0, v2 计算 v0, v1 / v2, v3 并将结果存入 v0, v1。
BF	rem-long/2addr vx, vy	计算 vx, vx+1 % vy, vy+1 并将结果存入 vx, vx+1 ^{注1} 。	BF20 - rem-long/2addr v0, v2 计算 v0, v1 % v2, v3 并将结果存入 v0, v1。
C0	and-long/2addr vx, vy	计算 vx, vx+1 与 vy, vy+1 并将结果存入 vx, vx+1 ^{注1} 。	C020 - and-long/2addr v0, v2 计算 v0, v1 与 v2, v3 并将结果存入 v0, v1。
C1	or-long/2addr vx, vy	计算 vx, vx+1 或 vy, vy+1 并将结果存入 vx, vx+1 ^{注1} 。	C120 - or-long/2addr v0, v2 计算 v0, v1 或 v2, v3 并将结果存入 v0, v1。
C2	xor-long/2addr vx, vy	计算 vx, vx+1 异或 vy, vy+1 并将结果存入 vx, vx+1 ^{注1} 。	C220 - xor-long/2addr v0, v2 计算 v0, v1 异或 v2, v3 并将结果存入 v0, v1。
C3	shl-long/2addr vx, vy	左移 vx, vx+1, vy 指定移动的位置, 并将结果存入 vx, vx+1。	C320 - shl-long/2addr v0, v2 以 v2 指定的位置左移 v0, v1, 结果存入 v0, v1。
C4	shr-long/2addr vx, vy	右移 vx, vx+1, vy 指定移动的位置, 并将结果存入 vx, vx+1。	C420 - shr-long/2addr v0, v2 以 v2 指定的位置右移 v0, v1, 结果存入 v0, v1。
C5	ushr-long/2addr vx, vy	无符号右移 vx, vx+1, vy 指定移动的位置, 并将结果存入 vx, vx+1。	C520 - ushr-long/2addr v0, v2 以 v2 指定的位置无符号右移 v0, v1, 结果存入 v0, v1。
C6	add-float/2addr vx, vy	计算 vx + vy 并将结果存入 vx。	C640 - add-float/2addr v0, v4 计算 v0 + v4 并将结果存入 v0。
C7	sub-float/2addr vx, vy	计算 vx - vy 并将结果存入 vx。	C740 - sub-float/2addr v0, v4 计算 v0 - v4 并将结果存入 v0。
C8	mul-float/2addr vx, vy	计算 vx * vy 并将结果存入 vx。	C810 - mul-float/2addr v0, v1 计算 v0 * v1 并将结果存入 v0。
C9	div-float/2addr vx, vy	计算 vx / vy 并将结果存入 vx。	C910 - div-float/2addr v0, v1 计算 v0 / v1 并将结果存入 v0。
CA	rem-float/2addr vx, vy	计算 vx % vy 并将结果存入 vx。	CA10 - rem-float/2addr v0, v1 计算 v0 % v1 并将结果存入 v0。
CB	add-double/2addr vx, vy	计算 vx, vx+1 + vy, vy+1 并将结果存入 vx, vx+1 ^{注1} 。	CB70 - add-double/2addr v0, v7 计算 v0, v1 + v7, v8 并将结果存入 v0, v1。
CC	sub-double/2addr vx, vy	计算 vx, vx+1 - vy, vy+1 并将结果存入 vx, vx+1 ^{注1} 。	CC70 - sub-double/2addr v0, v7 计算 v0, v1 - v7, v8 并将结果存入 v0, v1。
CD	mul-double/2addr vx, vy	计算 vx, vx+1 * vy, vy+1 并将结果存入 vx, vx+1 ^{注1} 。	CD20 - mul-double/2addr v0, v2 计算 v0, v1 * v2, v3 并将结果存入 v0, v1。
CE	div-double/2addr vx, vy	计算 vx, vx+1 / vy, vy+1 并将结果存入 vx, vx+1 ^{注1} 。	CE20 - div-double/2addr v0, v2 计算 v0, v1 / v2, v3 并将结果存入 v0, v1。
CF	rem-double/2addr vx, vy	计算 vx, vx+1 % vy, vy+1 并将结果存入 vx, vx+1 ^{注1} 。	CF20 - rem-double/2addr v0, v2 计算 v0, v1 % v2, v3 并将结果存入 v0, v1。
D0	add-int/lit16 vx, vy, lit16	计算 vy + lit16 并将结果存入 vx。	D001 D204 - add-int/lit16 v1, v0, #int 1234 // #04d2 计算 v0 + 1234 并将结果存入 v1。
D1	sub-int/lit16	计算 vy - lit16 并将结果存入 v	D101 D204 - sub-int/lit16 v1, v0, #int

	vx, vy, lit16	x。	1234 // #04d2 计算 v0 - 1234 并将结果存入 v1。
D2	mul-int/lit16 vx, vy, lit16	计算 vy * lit16 并将结果存入 v x。	D201 D204 - mul-int/lit16 v1, v0, #int 1234 // #04d2 计算 v0 * 1234 并将结果存入 v1。
D3	div-int/lit16 vx, vy, lit16	计算 vy / lit16 并将结果存入 v x。	D301 D204 - div-int/lit16 v1, v0, #int 1234 // #04d2 计算 v0 / 1234 并将结果存入 v1。
D4	rem-int/lit16 vx, vy, lit16	计算 vy % lit16 并将结果存入 v x。	D401 D204 - rem-int/lit16 v1, v0, #int 1234 // #04d2 计算 v0 % 1234 并将结果存入 v1。
D5	and-int/lit16 vx, vy, lit16	计算 vy 与 lit16 并将结果存入 vx。	D501 D204 - and-int/lit16 v1, v0, #int 1234 // #04d2 计算 v0 与 1234 并将结果存入 v1。
D6	or-int/lit16 v x, vy, lit16	计算 vy 或 lit16 并将结果存入 vx。	D601 D204 - or-int/lit16 v1, v0, #int 1234 // #04d2 计算 v0 或 1234 并将结果存入 v1。
D7	xor-int/lit16 vx, vy, lit16	计算 vy 异或 lit16 并将结果存 入 vx。	D701 D204 - xor-int/lit16 v1, v0, #int 1234 // #04d2 计算 v0 异或 1234 并将结果存入 v1。
D8	add-int/lit8 v x, vy, lit8	计算 vy + lit8 并将结果存入 v x。	D800 0201 - add-int/lit8 v0,v2, #int1 计算 v2 + 1 并将结果存入 v0。
D9	sub-int/lit8 v x, vy, lit8	计算 vy - lit8 并将结果存入 v x。	D900 0201 - sub-int/lit8 v0,v2, #int1 计算 v2 - 1 并将结果存入 v0。
DA	mul-int/lit8 v x, vy, lit8	计算 vy * lit8 并将结果存入 v x。	DA00 0002 - mul-int/lit8 v0,v0, #int2 计算 v0 * 2 并将结果存入 v0。
DB	div-int/lit8 v x, vy, lit8	计算 vy / lit8 并将结果存入 v x。	DB00 0203 - mul-int/lit8 v0,v2, #int3 计算 v2 / 3 并将结果存入 v0。
DC	rem-int/lit8 v x, vy, lit8	计算 vy % lit8 并将结果存入 v x。	DC00 0203 - rem-int/lit8 v0,v2, #int3 计算 v2 % 3 并将结果存入 v0。
DD	and-int/lit8 v x, vy, lit8	计算 vy 与 lit8 并将结果存入 v x。	DD00 0203 - and-int/lit8 v0,v2, #int3 计算 v2 与 3 并将结果存入 v0。
DE	or-int/lit8 v x, vy, lit8	计算 vy 或 lit8 并将结果存入 v x。	DE00 0203 - or-int/lit8 v0, v2, #int 3 计算 v2 或 3 并将结果存入 v0。
DF	xor-int/lit8 v x, vy, lit8	计算 vy 异或 lit8 并将结果存入 v x。	DF00 0203 0008: xor-int/lit8 v0, v2, #int 3 计算 v2 异或 3 并将结果存入 v0。
E0	shl-int/lit8 v x, vy, lit8	左移 vy, lit8 指定移动的位置, 并将结果存入 vx。	E001 0001 - shl-int/lit8 v1, v0, #int 1 将 v0 左移 1 位, 结果存入 v1。
E1	shr-int/lit8 v x, vy, lit8	右移 vy, lit8 指定移动的位置, 并将结果存入 vx。	E101 0001 - shr-int/lit8 v1, v0, #int 1 将 v0 右移 1 位, 结果存入 v1。
E2	ushr-int/lit8 vx, vy, lit8	无符号右移 vy, lit8 指定移动的 位置, 并将结果存入 vx。	E201 0001 - ushr-int/lit8 v1, v0, #int 1

			将 v0 无符号右移 1 位，结果存入 v1。
E3	unused_E3	未使用	
E4	unused_E4	未使用	
E5	unused_E5	未使用	
E6	unused_E6	未使用	
E7	unused_E7	未使用	
E8	unused_E8	未使用	
E9	unused_E9	未使用	
EA	unused_EA	未使用	
EB	unused_EB	未使用	
EC	unused_EC	未使用	
ED	unused_ED	未使用	
EE	execute-inline { 参数 }, 内联 ID	根据内联 ID ^{注6} 执行内联方法。	EE20 0300 0100 - execute-inline {v1, v0}, inline #0003 执行内联方法#3, 参数 v1,v0, 其中参数 v1 为"this"的实例, v0 是方法的参数。
EF	unused_EF	未使用	
F0	invoke-direct-empty	用于空方法的占位符,如 Object.<init>。这相当于正常执行了 nop 指令 ^{注6} 。	F010 F608 0000 - invoke-direct-empty {v0}, Ljava/lang/Object;<init>:()V // method@08f6 替代空方法 java/lang/Object;<init>。
F1	unused_F1	未使用	
F2	iget-quick vx, vy, 偏移量	获取 vy 寄存器中实例指向+偏移位置的数据区的值, 存入 vx ^{注6} 。	F221 1000 - iget-quick v1, v2, [obj+0010] 获取 v2 寄存器中的实例指向+10H 位置的数据区的值, 存入 v1。
F3	iget-wide-quick vx, vy, 偏移量	获取 vy 寄存器中实例指向+偏移位置的数据区的值, 存入 vx,vx+1 ^{注6} 。	F364 3001 - iget-wide-quick v4, v6, [obj+0130] 获取 v6 寄存器中的实例指向+130H 位置的数据区的值, 存入 v4,v5。
F4	iget-object-quick vx, vy, 偏移量	获取 vy 寄存器中实例指向+偏移位置的数据区的对象引用, 存入 vx ^{注6} 。	F431 0C00 - iget-object-quick v1, v3, [obj+000c] 获取 v3 寄存器中的实例指向+0CH 位置的数据区的对象引用, 存入 v1。
F5	iput-quick vx, vy, 偏移量	将 vx 寄存器中的值存入 vy 寄存器中的实例指向+偏移位置的数据区 ^{注6} 。	F521 1000 - iput-quick v1, v2, [obj+0010] 将 v1 寄存器中的值存入 v2 寄存器中的实例指向+10H 位置的数据区。
F6	iput-wide-quick vx, vy, 偏移量	将 vx,vx+1 寄存器中的值存入 vy 寄存器中的实例指向+偏移位置的数据区 ^{注6} 。	F652 7001 - iput-wide-quick v2, v5, [obj+0170] 将 v2,v3 寄存器中的值存入 v5 寄存器中的实例指向+170H 位置的数据区。
F7	iput-object-quick vx, vy, 偏移量	将 vx 寄存器中的对象引用存入 vy 寄存器中的实例指向+偏移位置	F701 4C00 - iput-object-quick v1, v0, [obj+004c]

	移量	的数据区 ^{注6} 。	将 v1 寄存器中的对象引用存入 v0 寄存器中的实例指向+4CH 位置的数据区。
F8	invoke-virtual-quick {参数}, 虚拟表偏移量	调用虚拟方法, 使用目标对象虚拟表 ^{注6} 。	F820 B800 CF00 - invoke-virtual-quick {v15, v12}, vtable #00b8 调用虚拟方法, 目标对象的实例指向位于 v15 寄存器, 方法位于虚拟表#B8 条目, 方法所需的参数位于 v12。
F9	invoke-virtual-quick/range {参数范围}, 虚拟表偏移量	调用虚拟方法, 使用目标对象虚拟表 ^{注6} 。	F906 1800 0000 - invoke-virtual-quick/range {v0..v5}, vtable #0018 调用虚拟方法, 目标对象的实例指向位于 v0 寄存器, 方法位于虚拟表#18H 条目, 方法所需的参数位于 v1..v5。
FA	invoke-super-quick {参数}, 虚拟表偏移量	调用父类虚拟方法, 使用目标对象的直接父类的虚拟表 ^{注6} 。	FA40 8100 3254 - invoke-super-quick {v2, v3, v4, v5}, vtable #0081 调用父类虚拟方法, 目标对象的实例指向位于 v2 寄存器, 方法位于虚拟表#81H 条目, 方法所需的参数位于 v3,v4,v5。
FB	invoke-super-quick/range {参数范围}, 虚拟表偏移量	调用父类虚拟方法, 使用目标对象的直接父类的虚拟表 ^{注6} 。	F906 1B00 0000 - invoke-super-quick/range {v0..v5}, vtable #001b 调用父类虚拟方法, 目标对象的实例指向位于 v0 寄存器, 方法位于虚拟表#1B 条目, 方法所需的参数位于 v1..v5。
FC	unused_FC	未使用	
FD	unused_FD	未使用	
FE	unused_FE	未使用	
FF	unused_FF	未使用	

注1: Double 和 long 值占用两个寄存器。(例: 在 vy 地址上的值位于 vy,vy+1 寄存器)

注2: 偏移量可以是正或负, 从指令起始字节起计算偏移量。偏移量在 (2 字节每 1 偏移量递增/递减) 时解释执行。负偏移量用二进制补码格式存储。偏移量当前位置是指令起始字节。

注3: 比较操作, 如果第一个操作数大于第二个操作数返回正值; 如果两者相等, 返回 0; 如果第一个操作数小于第二个操作数, 返回负值。

注4: 正常使用没见到过的, 从 [Android opcode constant list](#) 引入。

注5: 调用参数表的编译比较诡异。如果参数的数量大于 4 并且%4=1, 第 5 (第 9 或其他%4=1 的) 个参数将编译在指令字节的下一个字节的 4 个最低位。奇怪的是, 有一种情况不使用这种编译: 方法有 4 个参数但用于编译单一参数, 指令字节的下一个字节的 4 个最低位空置, 将会编译为 40 而不是 04。

注6: 这是一个不安全的指令, 仅适用于 ODEX 文件。